

NAVAL POSTGRADUATE SCHOOL

Monterey, California



19980428 049

THESIS

**PHOENIX AUTONOMOUS UNDERWATER VEHICLE (AUV):
NETWORKED CONTROL OF MULTIPLE ANALOG AND
DIGITAL DEVICES USING LONTALK**

by

Forrest C. Young

December 1997

Thesis Advisors:

Xiaoping Yun
Don Brutzman

Approved for public release; distribution is unlimited.

DTIC QUALITY INSPECTED 3

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE December 1997	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE PHOENIX AUTONOMOUS UNDERWATER VEHICLE (AUV): NETWORKED CONTROL OF MULTIPLE ANALOG AND DIGITAL DEVICES USING LONTALK			5. FUNDING NUMBERS	
6. AUTHOR(S) Young, Forrest C.				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT <p>The purpose of this thesis is to simplify analog and digital device control inside the <i>Phoenix</i> autonomous underwater vehicle (AUV). <i>Phoenix</i> is required to process many data information streams associated with a variety of different sensors. Real-time processing is required both for input sensing and for output directing. As presently configured, hardware devices aboard the <i>Phoenix</i> are manually connected and configured using parallel ports, serial ports, analog-to-digital (A/D) and digital-to-analog (D/A) controller hardware. Current hardware control within <i>Phoenix</i> connects all devices individually to a single computer. This approach is cumbersome, error-prone and does not scale.</p> <p>This project investigates the feasibility of using Echelon LonWorks hardware and LonTalk protocol as a faster and scalable networked robot control system. LonWorks/LonTalk is a flexible A/D D/A hardware networking technology that provides reliable communication, decentralized topology with no single point of failure, easy extensibility, excellent throughput, and interoperability for a wide variety of hardware.</p> <p>This project builds and tests a prototype LonTalk network that connects all <i>Phoenix</i> devices. This network demonstrates the capability of using LonWorks to control various types of hardware and support rapid component integration onboard the <i>Phoenix</i>. Successful demonstration of a LonTalk solution eliminates a critical barrier to <i>Phoenix</i> progress and makes robot execution much more robust.</p>				
14. SUBJECT TERMS Autonomous Underwater Vehicle, AUV, Networked Control, LonWorks Technology, LonTalk, LonBuilder			15. NUMBER OF PAGES 113	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

NSN 7540-01-280-5500

DTIC QUALITY INSPECTED 3

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

Approved for public release; distribution is unlimited

**PHOENIX AUTONOMOUS UNDERWATER VEHICLE (AUV): NETWORKED
CONTROL OF MULTIPLE ANALOG AND DIGITAL DEVICES USING
LONTALK**

Forrest C. Young
Lieutenant, United States Navy
B.S., University of California at Berkeley, 1990


Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING


from the


**NAVAL POSTGRADUATE SCHOOL
December 1997**

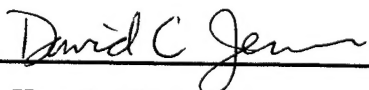
Author:


Forrest C. Young

Approved by:


Xiaoping Yun, Thesis Advisor


Don Brutzman, Thesis Advisor


for Herschel H. Loomis, Jr., Chairman
Department of Electrical and Computer Engineering

ABSTRACT

The purpose of this thesis is to simplify analog and digital device control inside the *Phoenix* autonomous underwater vehicle (AUV). *Phoenix* is required to process many data information streams associated with a variety of different sensors. Real-time processing is required both for input sensing and for output directing. As presently configured, hardware devices aboard the *Phoenix* are manually connected and configured using parallel ports, serial ports, analog-to-digital (A/D) and digital-to-analog (D/A) controller hardware. Current hardware control within *Phoenix* connects all devices individually to a single computer. This approach is cumbersome, error-prone and does not scale.

This project investigates the feasibility of using Echelon LonWorks hardware and LonTalk protocol as a faster and scalable networked robot control system. LonWorks/LonTalk is a flexible A/D D/A hardware networking technology that provides reliable communication, decentralized topology with no single point of failure, easy extensibility, excellent throughput, and interoperability for a wide variety of hardware.

This project builds and tests a prototype LonTalk network that connects all *Phoenix* devices. This network demonstrates the capability of using LonWorks to control various types of hardware and support rapid component integration onboard the *Phoenix*. Successful demonstration of a LonTalk solution eliminates a critical barrier to *Phoenix* progress and makes robot execution much more robust.

TABLE OF CONTENTS

I. INTRODUCTION: MINE WARFARE AND PHOENIX AUV.....	1
A. BACKGROUND.....	1
B. THESIS MOTIVATION AND GOALS.....	5
C. THESIS ORGANIZATION.....	5
II. PREVIOUS WORK.....	7
A. INTRODUCTION.....	7
B. AUV DESCRIPTION.....	7
C. RATIONAL BEHAVIOR MODEL (RBM) ARCHITECTURE.....	8
D. VIRTUAL AUV SOFTWARE DEVELOPMENT.....	10
E. BASIC STAMPS.....	10
F. RELATED PHOENIX WORK.....	11
G. SUMMARY.....	11
III. PROBLEM STATEMENT.....	13
A. INTRODUCTION.....	13
B. 10 HZ DATA PROCESSING RATE.....	13
C. COMPLEXITY OF CENTRAL ARCHITECTURE NETWORK.....	13
D. ADDING OR REMOVING DEVICES IN THE NPS AUV.....	14
E. FEASIBILITY OF REAL-TIME RESPONSE.....	14
F. SUMMARY.....	14
IV. CONTROL NETWORK: FOUNDATION AND COMPONENTS.....	15
A. INTRODUCTION.....	15
B. BACKGROUND.....	15
C. CONTROL NETWORK AND LONWORKS TECHNOLOGY.....	16
1. Control Network.....	16

2. LonWorks Technology.....	16
D. CONTROL NETWORK COMPONENTS.....	18
1. Neuron Chips and Application Nodes.....	18
2. Neuron-Chip Based Application Node.....	22
3. Communication Media.....	23
4. Connective Devices.....	23
5. Development Tools.....	26
6. Network Services Tools.....	27
E. SUMMARY.....	27
V. PROTOCOL AND NETWORK STRUCTURE OF LONWORKS.....	29
A. INTRODUCTION.....	29
B. LONTALK PROTOCOL.....	29
1. LonTalk Protocol.....	29
2. ISO/OSI Model.....	29
3. LonTalk Addressing Schemes.....	31
4. LonTalk Messaging Services.....	32
C. DATA COMMUNICATION MODES.....	33
D. LONWORKS NETWORK VARIABLES.....	37
E. LONWORKS PROGRAMMING MODEL.....	38
F. SUMMARY.....	38
VI. HARDWARE IMPLEMENTATION.....	39
A. INTRODUCTION.....	39
B. THE LONBUILDER DEVELOPER'S KIT.....	43
1. LonBuilder Development Station Enclosure.....	43
2. LonBuilder Interface Adapter.....	43
3. LonBuilder Control Processor.....	46
4. LonBuilder Neuron Emulator.....	48
5. LonBuilder SMX Adapter.....	50

6. LonBuilder Application Interface Kit.....	50
C. LONWORKS NEURON NODES.....	51
D. INPUT/OUTPUT DEVICES.....	57
E. THE NPS AUV'S NETWORK CONNECTION AND IMPLEMENTATION.....	58
F. LONWORKS NETWORK'S UPGRADE, MAINTENANCE, AND REPAIR.....	60
G. SUMMARY.....	61
VII. SOFTWARE METHODOLOGY.....	63
A. INTRODUCTION.....	63
B. LONBUILDER DEVELOPMENT ENVIRONMENT.....	64
1. State the Problem.....	64
2. Identify Nodes and Assign their Functions.....	65
3. Define the Interface of the External Device for Each Node	65
4. Write the Application Program for Each Node.....	66
5. Build, Debug, and Test Individual Nodes.....	70
6. Integrate Nodes into Networks and Test.....	71
C. SOURCE CODE DEVELOPMENT FOR THE APPLICATION NODES....	71
D. SOURCE CODE DEVELOPMENT FOR THE INTEGRATED NETWORK ONBOARD AUV.....	72
E. SUMMARY.....	74
VIII. CONCLUSIONS AND RECOMMENDATIONS.....	75
A. INTRODUCTION.....	75
B. RESEARCH CONCLUSIONS.....	75
1. Data Processing Rate Improvement.....	75
2. Network Architecture Simplification.....	75
3. No System Reconfiguration when Adding or Removing Devices.....	76
4. Real-Time Response.....	76

5. Suitability for Other Robot Architectures.....	76
C. RECOMMENDATIONS FOR FUTURE WORK.....	77
1. Implement LonWorks to NPS AUV.....	77
2. TCP/IP-to-LonTalk Telemetry Bridge.....	77
D. SUMMARY.....	78
APPENDIX A – MASTER SNVTs LIST.....	79
APPENDIX B – ESTIMATED COST FOR THE NPS AUV USING LONWORKS TECHNOLOGY.....	83
APPENDIX C – SOURCE CODE FOR THE AUV MANEUVERING EXERCISE.....	85
LIST OF REFERENCES.....	93
INITIAL DISTRIBUTION LIST.....	95

LIST OF FIGURES

Figure 1.1	External Components of the NPS AUV.....	2
Figure 1.2	Major Internal Components of the NPS AUV.....	3
Figure 1.3	Perspective View of the NPS AUV.....	4
Figure 2.1	RBM Tri-level Software Architecture for the Control of AUV.....	9
Figure 2.2	NPS AUV in a Virtual Environment.....	12
Figure 4.1	Neuron Chip Plan View.....	18
Figure 4.2	Function Blocks of Neuron Chip.....	19
Figure 4.3	Shared Memory Buffers with Three CPUs.....	20
Figure 4.4	Memory Maps of the Neuron Chip.....	22
Figure 4.5	Basic Structure of Free and Bus Topology.....	24
Figure 5.1	Differential Manchester Coding Scheme.....	35
Figure 5.2	Single-ended Mode Data Format.....	35
Figure 5.3	Differential Mode Data Format.....	36
Figure 5.4	Typical Packet Size for LonWorks Network.....	37
Figure 6.1	Overall Networked Control System Block Diagram in the NPS AUV.....	40
Figure 6.2	Picture of Networked Control System, Close View.....	41
Figure 6.3	Picture of Networked Control System, Far View.....	42
Figure 6.4	LonBuilder Developer's Kit.....	44
Figure 6.5	Block Diagram of LonBuilder Interface Adapter.....	45
Figure 6.6	Picture of Control Processor Board.....	47

Figure 6.7	Block Diagram of LonBuilder Control Processor.....	48
Figure 6.8	Picture of LonBuilder Emulator.....	49
Figure 6.9	Block Diagram of LonBuilder Emulator.....	50
Figure 6.10	Picture of LonBuilder Application Kit.....	51
Figure 6.11	Picture of IEC Flexible I/O Node.....	52
Figure 6.12	Block Diagram of IEC Flexible I/O Node.....	53
Figure 6.13	Picture of EMUP Burner.....	54
Figure 6.14	Picture of Serial to LonTalk Adapter (SLTA/2).....	56
Figure 6.15	Picture of Advance Motion Controls (ACS) Servo Amplifier.....	57
Figure 7.1	Integrated Development Environment.....	63
Figure 7.2	Neuron Chip I/O Pin Declarations.....	67
Figure 7.3	Neuron Nodes and Associated Network Variables.....	69

LIST OF TABLES

Table 4.1	Technical Data of Neuron Transceivers.....	25
Table 5.1	Seven Layers of ISO/OSI Model.....	31
Table 5.2	Neuron Chip's Data Rate over the LonWorks Network.....	34
Table 6.1	Typical Input/Output Address Assignment in a PC.....	46
Table 7.1	Neuron Node Names and their External Devices.....	64
Table 7.2	Functions of Neuron Nodes.....	66
Table 7.3	Neuron Node Names and their Network Variables.....	68
Table 7.4	Time Frame for the AUV Maneuvering Exercise.....	74

LIST OF ACRONYMS

A/D	Analog to Digital
ACKD	Acknowledged
AMC	Advanced Motion Controls
ANSI	American National Standard Institute
AUV	Autonomous Underwater Vehicle
CD	Collision Detection
CMOS	Complementary Metal-Oxide Semiconductor
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSMA	Carrier Sense Multiple Access
D/A	Digital to Analog
EEPROM	Electrically Erasable Programmable Read Only Memory
FTT	Free Topology Transceiver
GPS	Global Positioning System
I/O	Input/Output
IEC	Intelligent Technologies Corporation
ISO	International Standard Organization
LAN	Local-Area Network
LNS	LonWorks Network Services
LON	Local Operating Network
LPT	Link Power Transceiver
MAC	Media Access Control

NPS	Naval Postgraduate School
NSI	Network Services Interface
NSS	Network Services Server
OSI	Open System Interconnects
PC	Personal Computer
PCLTA	Personal Computer to LonTalk Adapter
PCNSS	Personal Computer to Network Services Server
PSG	Programmable Serial Gateway
PWM	Pulse Width Modulation
RAM	Random Access Memory
RBM	Rational Behavior Mode
REQUEST	Request/Response
ROM	Read Only Memory
SLTA	Serial to LonTalk Adapter
SMX	Standard Modular Transceiver
SNVTs	Standard Network Variable Types
TCP/IP	Transport Control Protocol/Internet Protocol
TPT	Twisted-Pair Transceivers
UNACKD	Unacknowledged
VLSI	Very-Large-Scale Integration
WAN	Wide-Area Network
XF	Transceiver

I. INTRODUCTION

A. BACKGROUND: MINE WARFARE AND PHOENIX AUV

As the cold war came to the end, naval threats posed by third world countries became a major concern for the United States military forces. Most third world countries do not have advanced weapon systems or elite military forces when compared to the United States. Nevertheless, the mine warfare capabilities of these countries is notable for low cost and simple technologies, and continues to be a great threat to the U.S. military forces operating in littoral regions. Damages suffered by the USS Princeton (CG-59), the Samuel B. Roberts (FFG-58), and the Tripoli (LPH-10) were due to minefields (Boorda 95). Lost lives, injuries and ship damage result in enormous costs. Alternative technologies are needed to reduce the threat posed by these minefields.

Although unmanned robots now appear to have the capability to search and detect such minefields, low-cost versions of the technology have not been demonstrated. The Naval Postgraduate School (NPS) *Phoenix* Autonomous Underwater Vehicle (AUV) has been built to prove such low-cost robots are possible. It is designed primarily to support research in autonomous under water mine hunting. It is also configured to complete other tasks such as underwater survey, pollution monitoring and remote observation.

An AUV is a self-contained unmanned vehicle equipped with many sensors, actuators, and controllers. The NPS *Phoenix* AUV design includes various motor controllers, thruster controllers, sonar sensors, dive tracker acoustic navigation, GPS/DGPS system, gyro system, and detectors. Figure 1.1 shows the external components of the NPS AUV. Figure 1.2 shows the major internal components of the vehicle. Figure 1.3 shows a perspective view of the NPS AUV.

The current NPS AUV uses OS-9 real-time operating system for its control code and its computer system, running on a GESPAC 68030 microprocessor (Brutzman 98). The hardware configuration of the NPS AUV is an assortment of devices connected to a centralized computer. This configuration requires multiple interface cards and independent wires connecting the central computer system to each individual device. As more functions are added and improvements made to the NPS AUV, additional devices and wires will be required to be added onboard. As a result, the vehicle becomes more and more complicated to maintain and to troubleshoot.

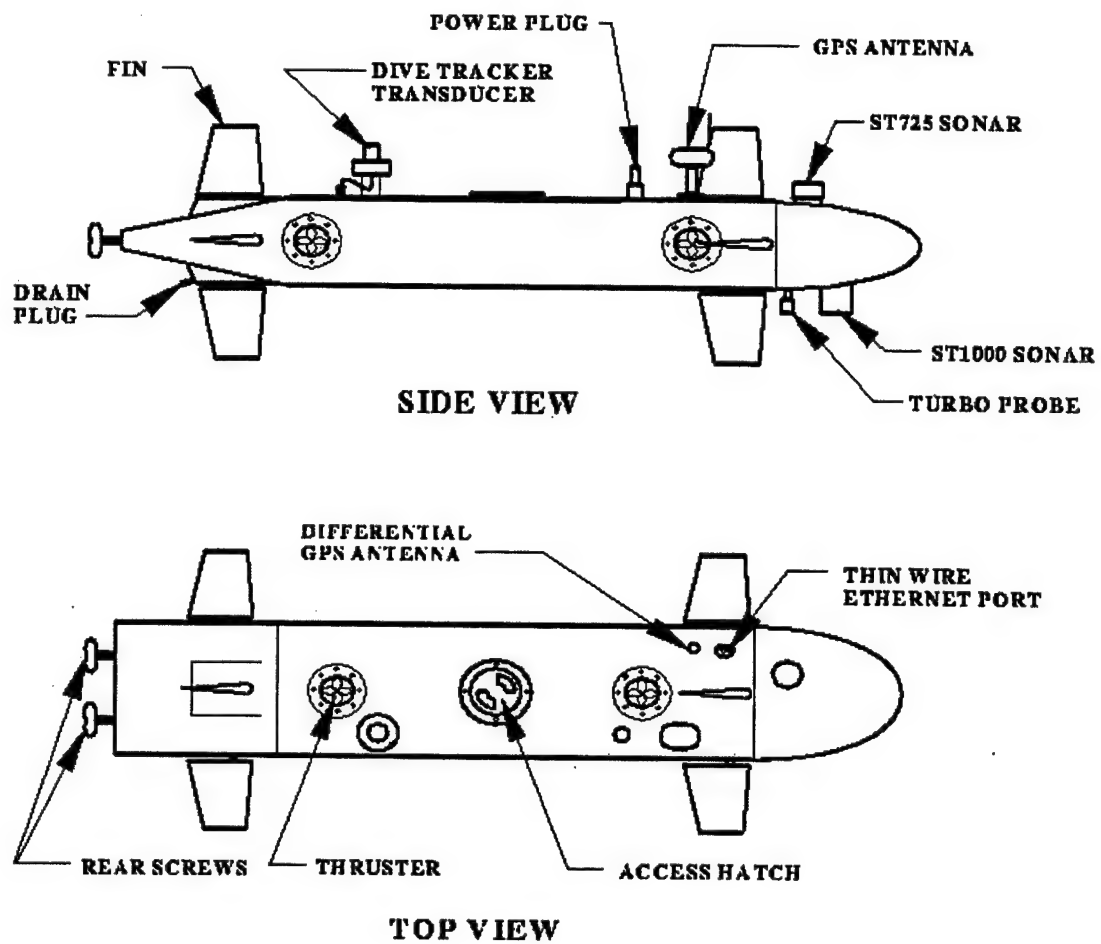


Figure 1.1 External Components of the NPS AUV (Marco 96)

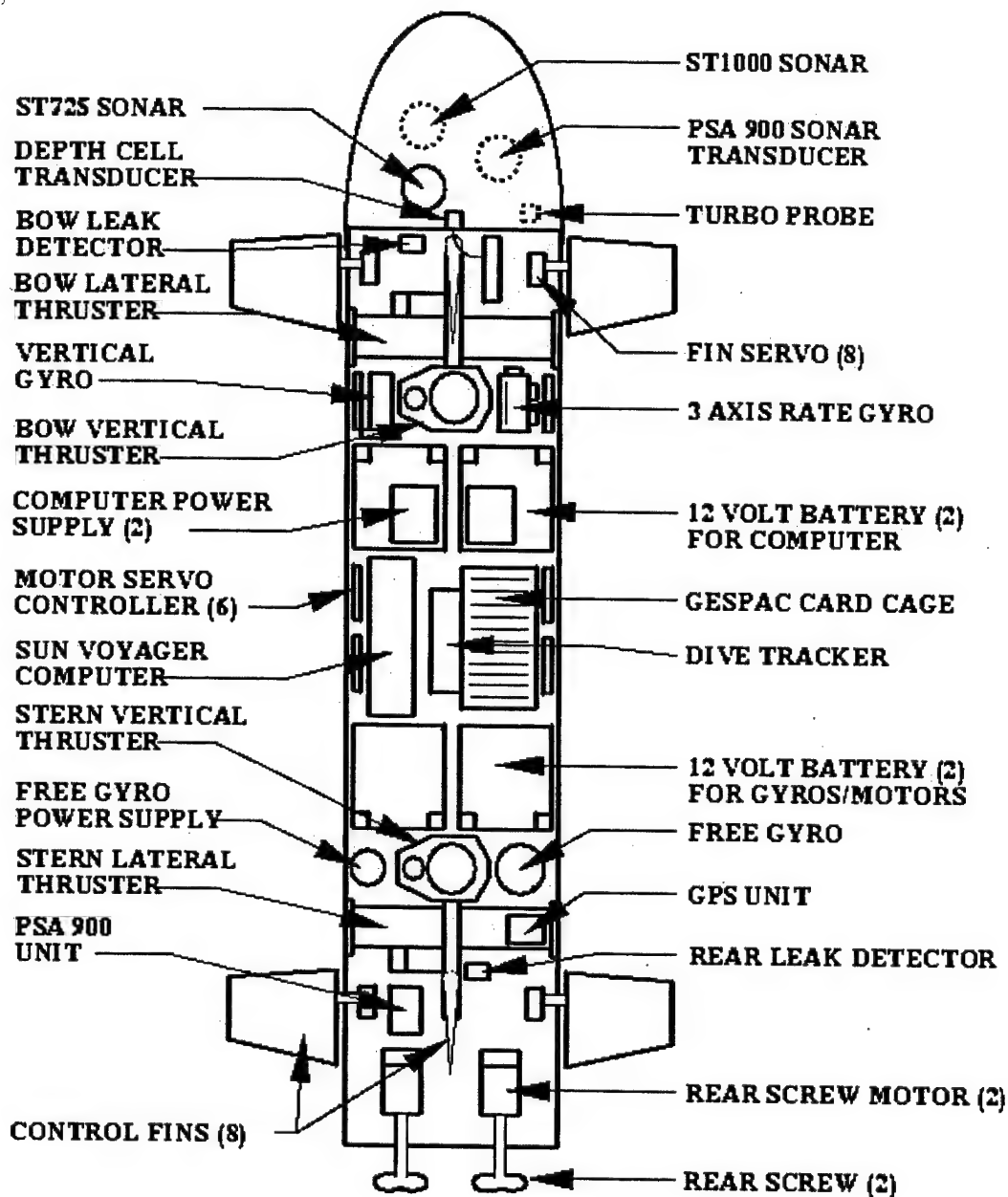


Figure 1.2 Major Internal Components of the NPS AUV (Marco 96)

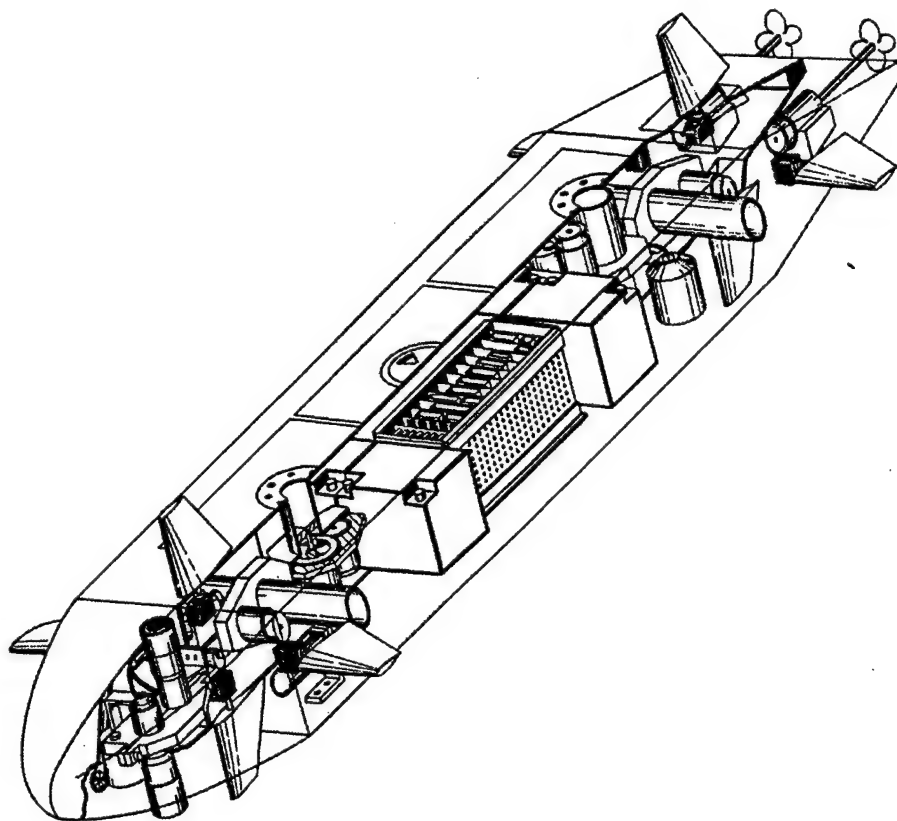


Figure 1.3 Perspective View of the NPS AUV (Marco 96)

B. THESIS MOTIVATION AND GOALS

The motivation for this project is to simplify the control of analog and digital devices within *Phoenix*. As presently configured, hardware devices onboard the NPS AUV are manually connected and configured using parallel ports, serial ports, analog-to-digital (A/D) and digital-to-analog (D/A) controller hardware. This approach is cumbersome, error-prone and does not scale. *Phoenix* is required to process many data information streams associated with many different onboard sensors, all within a very short period of time for both input sensing and output directing. The current approach to hardware control within the AUV is not satisfactory.

Echelon Lonworks hardware and LonTalk protocol is a flexible A/D and D/A hardware networking technology that appears to provide reliable communication, decentralized (peer-to-peer) topology with no single point of failure, easy extensibility and interoperability for a wide variety of hardware devices (Echelon 97). It appears that the reliability and throughput of *Phoenix* onboard sensors and effectors can be greatly improved using LonWorks system.

The primary goal of this thesis is to build and test a prototype LonWorks network that connects all *Phoenix* devices. The Echelon LonWorks development system is used to support rapid component integration, diagnosis and evaluation. If successful, this project will eliminate a critical barrier to *Phoenix* progress by making the execution level of the Rational Behavior Mode (RBM) (Byrnes 93) (Brutzman 98) software architecture much more robust. Finally, the thesis evaluates whether this approach is suitable as a general approach for other robot vehicles.

C. THESIS ORGANIZATION

The purpose of this thesis is to incorporate the LonWorks technology into the *Phoenix* in order to simplify analog and digital device control inside the vehicle. Chapter I presents the background, motivation and goals for this project. Chapter II reviews prior work which has significant relevance to this project, particularly the hardware configuration and software architecture of the system. Chapter III states in detail the problems addressed by this thesis. Chapter IV examines decentralized networked control

and summarizes the technology and components of LonWorks. Chapter V explains the protocol, network services and programming model of the LonWorks technology. Chapter VI describes the development tools, hardware construction and LonWorks employed network configuration. Chapter VII describes the software source code development to implement the LonTalk protocol for the networked control system within *Phoenix*. Chapter VIII provides thesis conclusions and presents recommendations for future research.

II. PREVIOUS WORK

A. INTRODUCTION

Much research has been conducted on *Phoenix* since 1987. *Phoenix* is an unmanned underwater vehicle designed for research in adaptive control, mission planning, mission execution, and post-mission data analysis (Healey 90).

This chapter summarizes the previous work conducted on *Phoenix*. A software architecture paradigm called the Rational Behavior Model (RBM) is written for the control of the NPS AUV system (Byrnes 93). A virtual world and computer simulation for the NPS AUV is created for speeding up the development of the NPS AUV (Brutzman 92) (Brutzman 95).

B. AUV DESCRIPTION

Physically, the *Phoenix* resembles a small-scale submarine. It has a cylindrical body shape, approximately 2.4 meters long, 0.46 meters wide and 0.31 meter deep. Externally, it has two aft propellers, two forward rudders, two aft rudders, two horizontal thrusters and two vertical thrusters to control its movement in the water. It is equipped with three different sonars: a Tritech ST 725 scanning sonar operating at 750 KHz (Tritech 92), a Tritech ST 1000 profiling sonar operating at 1250 MHz (Tritech 92), and a downward-looking altimeter. Additional devices include a depth cell for measuring the depth of AUV and a turbo-wheel probe for sensing water speed.

Internally, *Phoenix* has a GESPAC M68030 computer and Sun Voyager Sparc 5 Workstation (Brutzman 98). The GESPAC uses the OS-9 operating system for the real-time multitasking functions in the execution level for controlling the AUV's hydrodynamic stability (Byrnes 93). The GESPAC/OS-9 combination is a relatively slow computer system. The Sun Voyager 5 uses SunOS 5.4 for data storage in the tactical and strategic level. An Ethernet connects these two systems to form a Local-Area Network (LAN) inside the NPS AUV. This greatly simplifies remote monitoring and testing of *Phoenix* by providing Internet connectivity, either by cable connection or radio modem. A GPS system is installed for tracking the AUV's location via longitude and latitude. The gyro system is used for sensing the vehicle's orientation and angular rate

about three degrees of rotational freedom (Burns 96). The power supply for the internal electronic components is provided by multiple 24 volt batteries. Numerous A/D and D/A converters for computer-hardware interfaces are currently at maximum capacity, with insufficient connectivity to control the full number of devices aboard.

C. RATIONAL BEHAVIOR MODEL (RBM) ARCHITECTURE

The software architecture of the NPS AUV is a tri-level RBM architecture (Byrnes 93) (Brutzman 98). There are three levels in the model: the strategic level, tactical level, and execution level. Figure 2.1 shows these three levels and their interactions with each other.

The strategic level is the highest level, responsible for the overall operating condition of the NPS AUV. It prepares plans and makes operational decisions for the vehicle. This level interacts with the tactical level in order to obtain valuable information to determine current status of vehicle and the operating environment and provides guidance to the tactical level. There is no timing restriction or quantitative analysis in this level, therefore, the strategic level operates in an asynchronous environment.

The tactical level lies between the strategic and execution levels. It receives general guidance and objectives for a particular mission from the strategic level. It then issues various commands directly to the execution level to carry out the tasks necessary to accomplish the mission. The tactical level interacts with execution level via a message-passing protocol, and interacts with strategic level via function calls. The tactical level also operates in an asynchronous mode because it does not require interacting with the hardware in a hard-real-time manner (Byrnes 93).

The execution level is the lowest level in RBM. This level is written in the 'C' language. It interacts with the GESPAC operating system to issue all commands to various devices. This level is responsible for the control and stability of the *Phoenix*. It controls all the external devices, such as motors, thrusters, sonars, control planes and communications. It is required to handle these control tasks on a real-time basis for the safe operation of the vehicle. It has to operate in a synchronous mode. If any dangerous situations arise, such as flooding, low power supply, loss of communication or loss of

depth control, the execution level can override the strategic and tactical level and abort its mission to assure the overall safety of the vehicle.

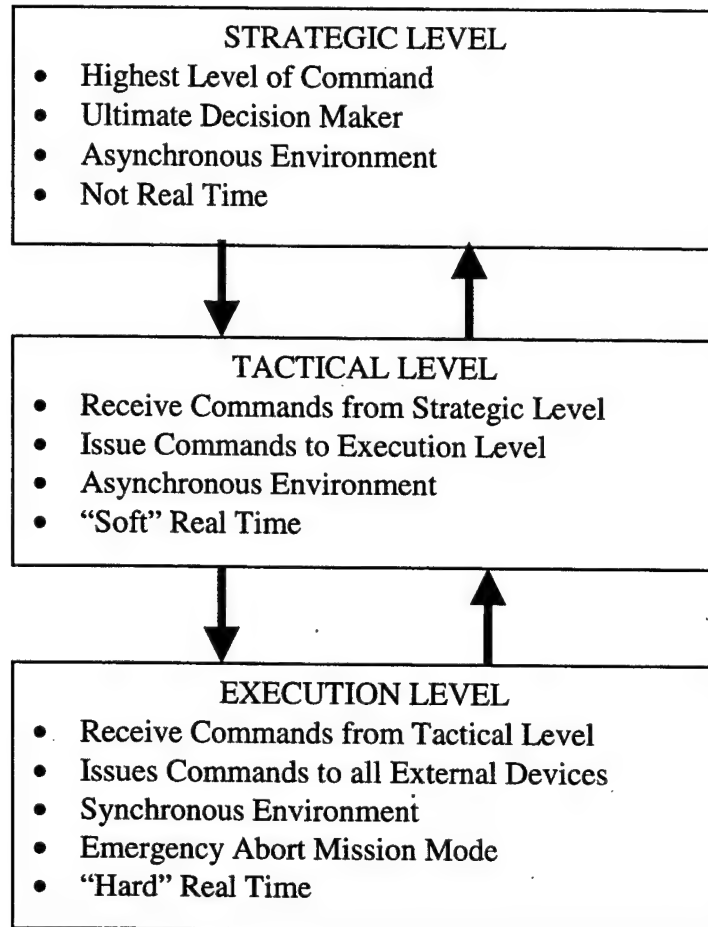


Figure 2.1 RBM Tri-Level Software Architecture for the Control of AUV

D. VIRTUAL AUV SOFTWARE DEVELOPMENT

Construction and development of the *Phoenix* have historically been slow but steady. With the advent of computer simulation and its application to this project, many problem areas which would normally be detected in actual in-water experiments can now be discovered during virtual environment tests. Many problem areas that could cause catastrophic failure during in-water testing can be safely detected and corrected through the use of computer simulation. Also, the use of such simulation greatly speeds the problem detection-to-correction cycle and reduces overall testing costs.

Once problem areas have been identified through the simulation, the source code can be corrected and the simulation run again to assess the alteration. When the AUV appears to operate correctly through the simulation and all source code corrections have been made, the source code on the *Phoenix* can be updated and in-water testing can begin. The simulator provides pre-mission testing, psuedo-mission testing, and post-mission playback. This virtual AUV environment has been created at NPS by Don Brutzman (Brutzman 94). Figure 2.2 depicts the virtual AUV in its computer generated environment.

In addition to a pure virtual simulation, the simulation can run with the *Phoenix* in-the-loop to more quickly update source code on the robot. This combination of virtual and physical models has greatly increased the pace of progress in project development. A detailed description of this merging of virtual and physical AUV control software is provided by Burns (Burns 96).

E. BASIC STAMPS

An alternative control network technology was also examined: BASIC Stamps (Parallax 97). These are small microcomputer chips that run Parallax BASIC (PBASIC) programs to directly interface with TTL-level devices via programmable I/O pins. Typical examples of these devices are LEDs, speakers, and shift registers. BASIC Stamps can also interact with non-TTL devices such as solenoids, RS-232 serial devices, and other hardware using a variety of adapters.

BASIC Stamps can control many different types of application nodes. The Stamps A/D converter application node provides the hardware and software required to interface an analog-to-digital converter to the Parallax BASIC Stamp. The Stamps servo

application node provides a program to control pulse-width proportional servos by using Parallax BASIC. The indoor sonar range-finding application node provides a circuit that allows the BASIC Stamp to measure distances for one to twelve feet using ultrasonic transducers.

BASIC Stamps technology is promising, but it still has some shortfalls for robot use. BASIC Stamps are single analog-to-digital conversion devices that are hard to network. Problems with multiple devices are thus difficult to locate and isolate. Each device connected to a Stamp also is likely to require an individual computer connection - an approach that does not scale. BASIC Stamps may be useful on rare occasions to interface specialized analog equipment to LonTalk Neuron nodes. BASIC Stamps do not fully solve the need by the *Phoenix* AUV for a fully networked hardware control system.

F. RELATED PHOENIX WORK

Despite the great effort and numerous hours of research and development that have been conducted to date, much remains to be done. The research associated with this thesis is just a small part of this greater ongoing effort to continually improve the *Phoenix* AUV. This thesis focuses on reconfiguring and simplifying the analog and digital device controls presently installed in the *Phoenix*. Other research being conducted to improve *Phoenix's* performance include: precise compass calibration by Xiaoping Yun and Randy Knapp, virtual NPS AUV hydrodynamics model refinement by Kevin Byrne, RBM Tactical Level formalization, refinement, and generalization by Michael Holden, and the use of 3-D graphics for sonar and tactical environment visualization by Timothy Holliday.

G. SUMMARY

During the development of the NPS *Phoenix* AUV, numerous sensors, actuators, sonars, and controllers have been installed onboard the vehicle. These devices are connected by using a centralized control networked system. The amount of wire and the configuration of this system are complex, cumbersome, and inefficiently unorganized. In this thesis, the concept of a decentralized peer-to-peer networked control system for analog-digital communications is examined. This thesis investigates the feasibility of

using Echelon LonWorks Technology to reorganize and simplify the hardware control system onboard of the NPS *Phoenix* AUV.

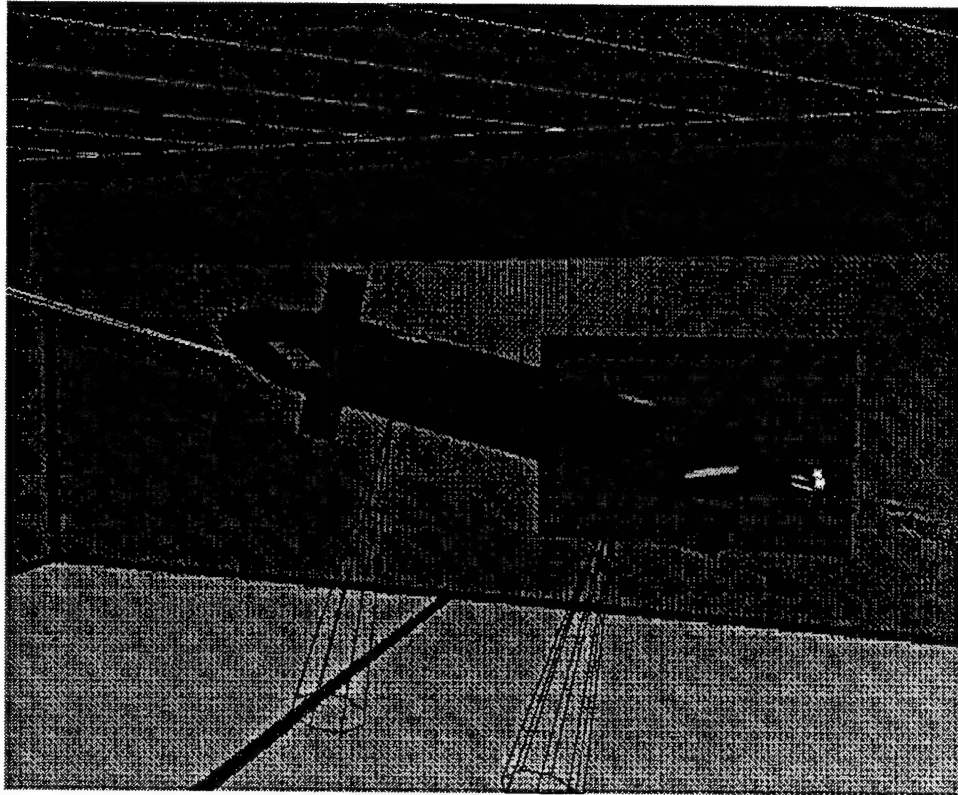


Figure 2.2 NPS AUV in Virtual Environment (Burns 96)

III. PROBLEM STATEMENT

A. INTRODUCTION

The increasing amount of information processed on the *Phoenix* has created a data processing and communication bottleneck. The current system configuration onboard *Phoenix* cannot provide adequate real-time response. This is a result of the 10 Hz maximum processing rate, the complex network control system and architecture, and the need to reconfigure the entire system if any component is added or removed. This chapter addresses each of these in turn.

B. 10 HZ DATA PROCESSING RATE

The current AUV configuration uses a central architecture system, a GESPEC computer system utilizing the OS-9 operating system, which uses a single central computer to interact with all *Phoenix* hardware components. As previously stated the maximum data processing rate is 10 Hz; this does not provide adequate bandwidth to control all of the devices onboard *Phoenix* and subsequently results in severe control problems.

This problem can be solved by using the LonWorks Technology with a 10 MHz Neuron processor and a different networked control architecture. This increases the potential aggregate bandwidth to 1.25 Mbits per second which will pass up to 1000 packet messages per second at peak load, and up to 800 packets per second continuously.

C. COMPLEXITY OF CENTRAL ARCHITECTURE NETWORK

The complexity of a centralized control system is cumbersome and inefficient. Each component (motor controller, servo amplifier, sonar, etc.) in *Phoenix* requires separate inputs into this central control system. The amount of wire and number of required connections combine to make this centralized networked control system extremely complicated and particularly difficult to troubleshoot.

This problem can be solved by using a decentralized (peer-to-peer) networked control system to simplify the configuration and increase the overall efficiency of the *Phoenix*.

D. ADDING OR REMOVING DEVICES IN THE NPS AUV

Presently, adding or removing any device from the *Phoenix* requires a change to the entire software configuration with no guarantee of driver compatibility between the existing component drivers and the new one. Troubleshooting these conflicts can be very difficult and frustrating.

This problem can be solved by connecting each device to a LonWorks networked Neuron node. Each Neuron node is independent and equipped with its own processor, thus it greatly simplifies the removal or addition of components without having to reconfigure the entire system.

E. FEASIBILITY OF REAL-TIME RESPONSE

The ultimate goal of the NPS AUV is to provide a real-time data analysis during its mission. The current configuration does not provide this feature. This project attempts to achieve real-time data analysis by using a Pentium 100 processor along with the aforementioned decentralized networked control system. The target performance is 10 Hz or better for each sense-decide-act loop which queries and commands all sensors and actuators in sequences.

F. SUMMARY

This chapter addressed the problems embedded in the current *Phoenix* using a central networked control system. This project is designed to improve the overall performance of the *Phoenix* and realizing the objective of real-time data analysis by implementing improvements in each of these problem areas.

IV. CONTROL NETWORK: FOUNDATION AND COMPONENTS

A. INTRODUCTION

The use of a control network is a new approach to machine control intended to improve the efficiency and reliability of many complex systems. The LonWorks approach to distributed control utilizes network technology with many independent Neuron nodes as its network foundation. Each Neuron node is an intelligent node that has a local processor that can receive, process and send out local data in cooperation with the entire network. This chapter discusses the basic ideas and individual components of LonWorks technology. Detailed information can be found in Echelon LonWorks technical manuals, listed in the references.

B. BACKGROUND

In the early stages of computer evolution, large mainframe computer systems dominated the industry. They contained centralized systems with a master and slave architecture. This architecture provided a single machine used by many users. As computer technology and performance improved, the size of microprocessors and peripherals shrank dramatically. Powerful personal desktop computers have flourished that can perform various tasks, such as word processing, spreadsheet calculation and computation-intensive graphics applications. Typically these personal computers are independent machines used only by one person, and do not share information with each other directly. An external device such as a floppy disk was, until relatively recently, the only tool for sharing data among computers. It is an inefficient way to share information.

Today's personal computers are more powerful and able to handle more tasks using a client/server network system. Wide-Area Networks (WANs) and Local-Area Networks (LANs) using TCP/IP are typical network systems with a decentralized architecture. Despite the distributed nature of TCP/IP, many systems are designed as client/server systems. A single point of failure in the central control server might lead to failure for all systems using that server, and thus can be considered unreliable. For small systems such as robots, a client/server central network system also requires additional wiring and hardware.

The concept of distributed control networks has been introduced in an effort to improve the current TCP/IP system. A control network connects smart nodes (nodes with their own microprocessors) that function independently, and communicate with each other. This allows nodes to implement sense and control applications locally. An independent node contains a local processor for control functions, a transceiver for media access and communication functions, an input/output interface to interact with I/O devices, and the necessary software code for operating system, protocol, and local library functions. A control network can increase overall system performance and avoid single points of failure by distributing the processing power to each individual node.

C. CONTROL NETWORK AND LONWORKS TECHNOLOGY

1. Control Network

A control network is a group of intelligent nodes that communicate with each other to implement input sensing and output directing functions. In an application such as building control management, the building thermometers and lighting switches are the input sensing devices. The air conditioning units and lighting systems respond to these sensing devices and act as output control devices. Applying control network to an application such as an automobile, the car's optical or radar detectors sense external obstacles, and the system issues control commands via the network to actuate the brake system and warn the driver.

Control networks use two types of communication: peer-to-peer (distributed control) and master-to-slave (centralized control). The network data processing loads are distributed among all nodes. The combined power of these distributed processors can increase the performance of a network system. The control functions can also be distributed in a peer-to-peer type of communication. It has no single point of failure, thus the reliability of a network system can be enhanced.

2. LonWorks Technology

A Local Operating Network (LON) consists of intelligent devices, or nodes, that are connected by one or more communications media and that communicate with one another using a common protocol. This technology allows a system to sense, process,

communicate, and control a system that is distributed over a network. This capability encompasses a multitude of applications. It allows products to be linked together and communicate to serve applications ranging from small instruments to large and complex process control systems (Echelon LonWorks Reference CD-ROM 97).

As embodied in the LonWorks system, LON consists of intelligent nodes, each containing a local operating processor, a Neuron chip. The nodes on a LON receive various inputs from external devices. Those inputs are processed and broadcast to the network by using the local node's Neuron processor. The nodes also respond to changes in the network to produce desired outputs. Each node can perform different functions. In most cases, a single node is designed to perform a very simple function. However, by grouping different nodes together, they cooperate to perform more complex tasks. These tasks can be in a broad spectrum of applications, depending on the types of nodes in the network.

A LON control network can range in size from two to 32,385 nodes in a domain, each with one or more sensors or actuators, plus localized computational capability. Sensors are used to collect data and information from external devices. Actuators are used to receive commands and control the external devices. The local processor is used to process local data, perform analysis and conversion, and then report any significant changes in its environment.

In a LonWorks system, the design and development of hardware, software, and network are all independent tasks. A node's function is only specified and programmed for its intended external device, a LonMark "object." LonMark objects describe standard formats for how information is input to and output from a node and shared with other nodes on the network (LonMark 96).

By simplifying the design of a single node and making it independent of most external influences, we can reduce development effort and cost. Nodes become generic building blocks that can be used and re-used in various environments. For example, a generic motor actuator node can be used to control propellers in various rotational speeds. In another application, it can control fin motors in a pulse width modulation mode, without any change to the application code or node hardware.

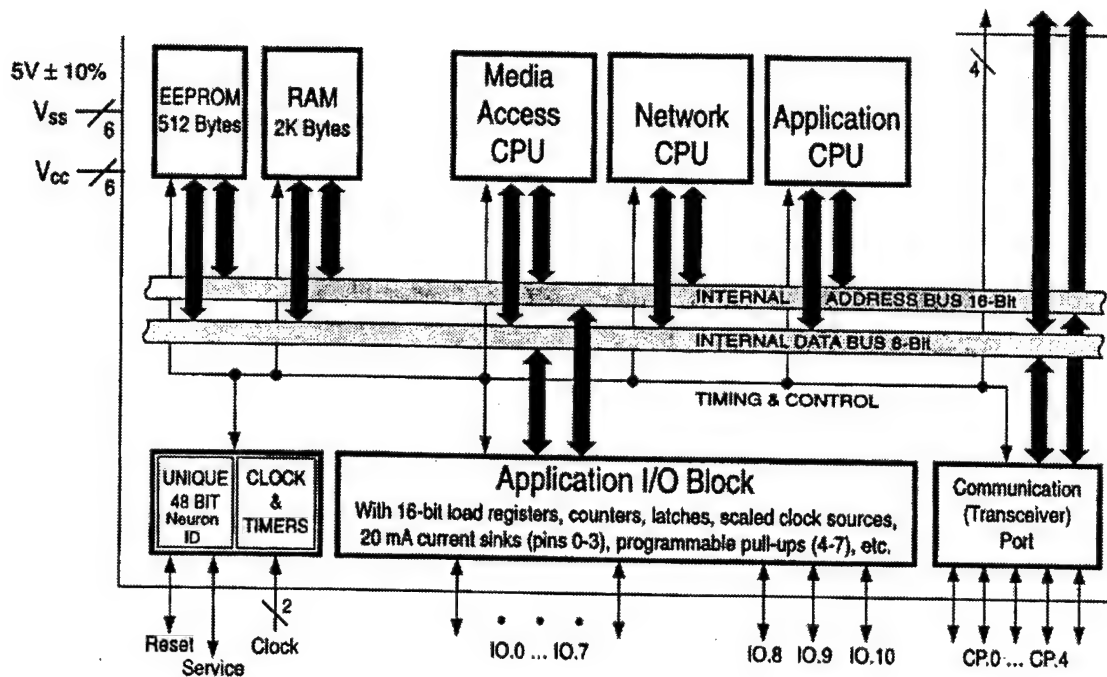


Figure 4.2 Function Blocks of Neuron Chip (Echelon, Neuron Chip Data Book)

The three 8-bit CPUs on each Neuron are identical and they can perform many different networking functions. Basically, they are divided into three areas: the Media Access Control (MAC), Network interface, and Application interface.

CPU-1 is the MAC CPU that handles layers one and two of the seven-layer LonTalk protocol stack. (Chapter V will describe the LonTalk Protocol in more detail). Its main function is to execute the user's application such as measuring input parameters, timing events, making logical decisions, and driving outputs. Its processing includes driving the communications subsystem hardware as well as executing the media access algorithm. It communicates with CPU-2 using network buffers located in shared memory. Figure 4.3 shows the shared memory buffers interact with all three CPUs. Access to them is mediated with hardware semaphores to resolve contention when updating shared data.

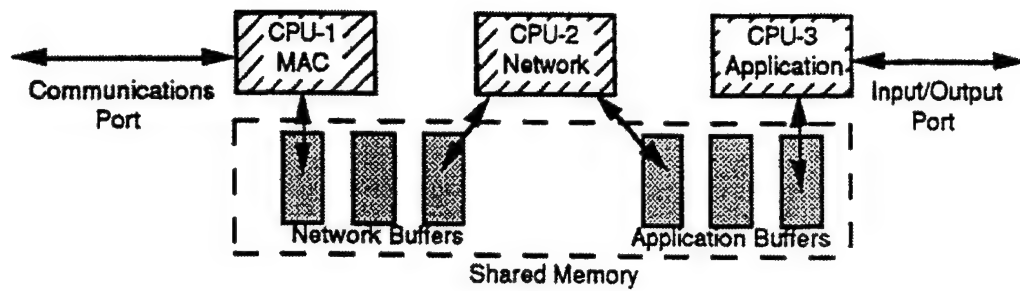


Figure 4.3 Shared Memory Buffers with Three CPUs (Echelon, Neuron Chip Data Book)

CPU-2 is the Network CPU that handles layers three through six of the LonTalk protocol stack. Its main function is encoding and decoding the messages to be sent over the network. It handles network variable processing, addressing, transaction processing, authentication, background diagnostics, software timers, and network management. It uses network buffers to communicate with CPU-1, and application buffers to communicate with CPU-3. The buffers are also located in shared memory.

CPU-3 is the Application CPU. Its main function is to control the Network Communication Port that physically sends and receives the packets of data. It runs code written by the user, together with the operating system services called by application code. The programming language used by the application programmer is Neuron C, a derivative of the ANSI C language modified for LonWorks distributed control applications. The major modifications include the following:

- A declarative syntax for input/output objects directly mapping into the input/output capabilities of the Neuron chip.
- A declarative syntax for network variables, which are Neuron C language objects whose values are automatically propagated over the network whenever values are assigned to them.
- A declarative syntax for millisecond and second timer objects which activate user tasks on expiration.

- A library of functions, which when called, can perform event checking, manage input/output activities, send and receive messages across the network, and control miscellaneous functions of the Neuron chip.

Each Neuron chip has the memory components of EEPROM, RAM, ROM, and external memory. The internal EEPROM of each Neuron chip contains the following information:

- (1) Network configuration and addressing information
- (2) 48-bit Neuron chip identification code
- (3) User-written application code and read-mostly data

User data in EEPROM can be written under program control. The Neuron chip uses an on-board charge pump to generate the required programming voltage. The charge pump operation is transparent to the user. The total erase and write time is 20ms per byte. The EEPROM may be written 10,000 times with no data loss.

The EEPROM of each Neuron chip stores installation-specific information such as network addresses and communication parameters. Each Neuron chip has a 48-bit identifier, the Neuron ID, that is permanently written into the EEPROM during manufacture. This 48-bit Neuron ID is unique; the possibility of two Neuron chips have the same identical ID is zero. This can eliminate any confusion and overlapping problems for any network connection.

The RAM of each Neuron chip is used to store:

- (1) Stack segment, application, and system data
- (2) LonTalk Protocol network buffers and application buffers

The RAM State is retained as long as power is applied to the chip, even in sleep mode. However, when the node is reset, the RAM is cleared.

The ROM of Neuron chip stores the Neuron chip firmware, including:

- (1) LonTalk protocol code
- (2) Event-driven task scheduler
- (3) Application function libraries

The Neuron 3150 chip uses external memory instead of on-chip ROM. This chip can support up to 59,392 bytes of addressing for the external memory. Application program data, the Neuron chip's firmware, and reserved space are stored in the external memory. Figure 4.4 shows the memory maps of the Neuron Chip.

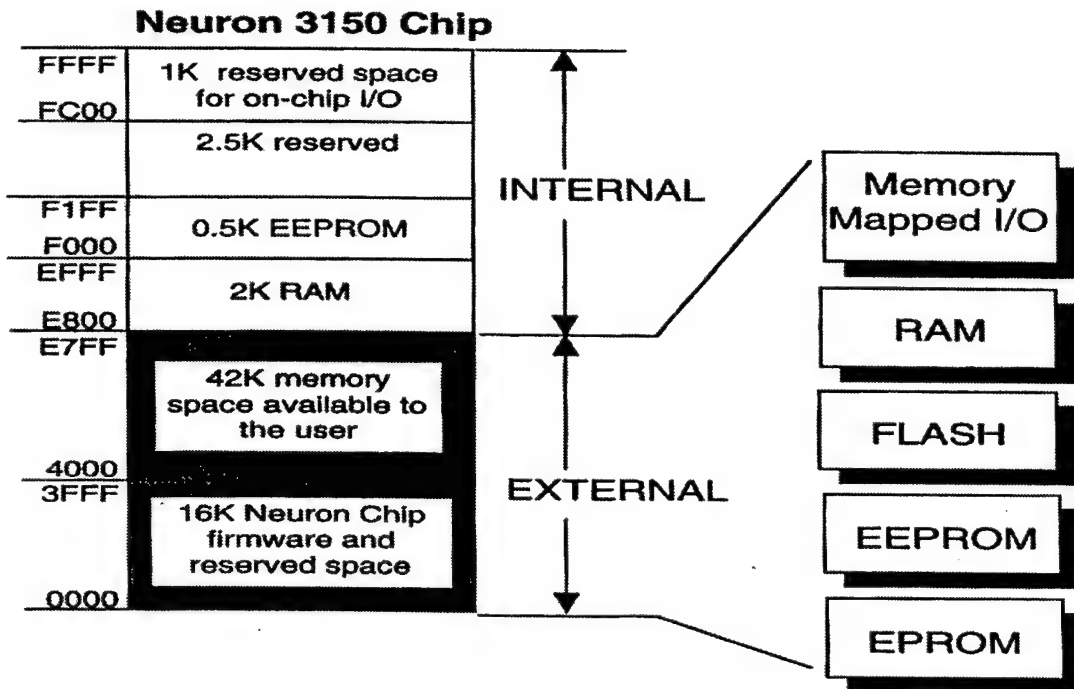


Figure 4.4 Memory Maps of the Neuron Chip (Echelon, Neuron Chip Data Book)

2. Neuron-Chip Based Application Node

An application node has the capability of receiving data from sensors, processing input data locally, and executing the desired control task. A single Neuron node contains the following components as the minimum requirements: 1) a Neuron Chip, 2) a transceiver, 3) circuitry to connect the Neuron chip to input/output devices, 4) an optional host processor.

The Neuron chip is a microcontroller for processing data locally, and implementing the LonTalk protocol. The transceiver is used to communicate input/output data with the network. Circuitry is built to connect the Neuron chip and its input/output devices. These input/output devices can be motor servos, motor controller, sensors, and actuators. The optional host processor is a processor other than the Neuron chip. It is primarily used to execute the node's application in network management and

maintenance, which required more processing power. Most LonWorks nodes use the Neuron chips as the local processor, and for simple input/output tasks they are adequate.

3. Communication Media

LonWorks uses two types of communication media. One is wireless communication with radio frequency or infrared. The other type is wired communication using media such as twisted pair cable, power line, or coaxial cable.

4. Connective Devices

The connective devices of LonWorks include various types of transceivers, control modules, routers, and network services interface.

The transceivers can be categorized into two groups depending on the networked topology. There are two types of topology used in LonWorks technology. The first one is Free Topology. It consists of devices connected to the communication channel in random multi-dropped fashion. There is only one termination box in this topology. The termination box is required for proper data transmission performance in the network segments. The termination box is used to absorb any signal and remove it from the network. The communication channel can have the configuration of a ring, star, bus, or mixed. The second one is Bus Topology. It consists of a central main communication channel. It is called the bus with two termination boxes, one at each end. Each device is attached through hardware interfacing, known as a node, to the communication channel in a multi-dropped fashion. The top box of Figure 4.5 shows a bus topology. Figure 4.5 bottom box shows different configurations of free topology (Echelon, Training Manual).

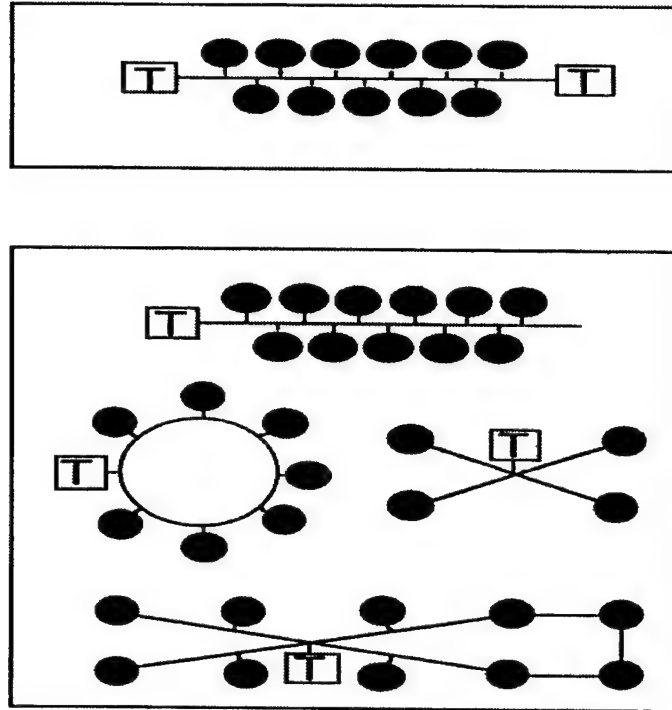


Figure 4.5 Basic Structure of Free and Bus Topology (Echelon, Training Manual)

There are two transceivers used in free topology. One is Link Power Transceiver (LPT-10), and the other one is Free Topology Transceiver (FTT-10). The LPT-10 combines power and data on a common twisted wire pair. It only requires one 48 Volt DC external power supply for the network. Power flows through the LPT-10 Link Power Interface Module and twisted pair lines to all the nodes. The transceiver integrated a +5 Volt DC regulator for all the nodes. It provides +5 Volts DC at up to 100 mA to all the nodes and eliminates a local power supply at each node. This voltage and current is high enough to power a Neuron Chip and associated components. Using this transceiver can eliminate local power supplies, resulting in equipment and labor cost saving (Echelon, LonWorks Products Manual).

The FTT-10 is compatible with LPT-10. They can communicate with each other in the same twisted pair medium. If a node with its associated devices requires higher voltage and current than the link power segment can provide, then it needs an additional local power supply. A node equipped with FTT-10 transceiver can be operated by a local power supply. It can communicate with all the nodes in the link power network without

any electrical isolation. It is another cost saving design (Echelon, LonWorks Products Manual).

These two transceivers can be used in any type of topology without any restriction. Table 4.1 describes the technical data for the transceivers.

Transceiver	Medium	Bit Rate	Topology	Distance	No. of Nodes
LPT-10	Twisted Pair	78 Kbps	Free-Bus, Star, Loop, Others, Combinations	500m free topology	32
FTT-10	Twisted Pair	78 Kbps	Free-Bus, Star, Loop, Others, Combinations	500m free topology	64
TPT/xf-78	Twisted Pair	78 Kbps	Bus	1400m (3m stubs)	64
TPT/XF-1250	Twisted Pair	1.25 Mbps	Bus	130m (0.3m stubs)	64

Table 4.1 Technical Data of Neuron Transceivers (Echelon, LonWorks Products Manual)

The Twisted Pair Transceivers, TPT/XF-78 and TPT/XF-125, are used in Bus Topology. Table 4.1 also shows their technical data. Each transceiver includes a transformer-isolated communication transceiver and connectors for power, the neuron Chip communication port lines, and the twisted pair bus. The TPT/XF-78 operates at 78kbps data transmission rate, and the TPT/XF-125 operates at 1.25Mbps. The TPT/XF-125 has the highest data transmission rate. It is the desired transceiver to meet the high speed networking application, like the NPS AUV (Echelon, LonWorks Products Manual).

LonWorks control modules integrate a Neuron Chip, communication transceiver, memory, and clock oscillator in one compact module. They require a power supply, the local sensors/actuators, and the application program running on the Neuron Chip in order to build a complete node (Echelon, LonWorks Products Manual).

When there are two or more different media in LonWorks network, a LonWorks Router can be used. It provides flexibility to the network system. The router can be used

to increase maximum number of nodes and total wire length in the network. It can also increase total system reliability by dividing network into many subsets.

If additional processing power is needed for a node, a host processor can be part of LonWorks node in addition to the local Neuron Chip. Echelon LonWorks offers parallel ISA bus interfaces, serial (EIA-232) interfaces, PC Card interfaces, and Hayes-compatible modem interfaces for these applications. The PCLTA PC LonTalk Adapter is a PC plug-in card that provides access to a LonWorks network from any ISA bus PC with a compatible operation system. The SLTA serial LonTalk Adapter is an integrated LonWorks network interface that can be used to interface any host equipped with EIA-232 serial interface to a twisted pair LonWorks network. The PSG/2 is a programmable gateway version of the SLTA/2 adapter. It provides more flexibility for the designer to create more versatile control systems or devices (Echelon, LonWorks Products Manual).

A LonWorks node with local host processor in addition to the Neuron Chip requires a network service interface to connect with LonWorks network. Such node usually performs more complex tasks, for example, monitor the entire system, record network data, and provide installation, maintenance, and diagnostic tools. This node can simultaneously support network communications and also support high-level interaction with a network tool. The Network Service Server (NSSs) and Network Service Interface (NSI) are designed to meet the above requirements (Echelon, LonWorks Products Manual).

5. Development Tools

LonWorks technology provides a development tool, the LonBuilder Developer Kit, for helping the developer to design and develop the LonWorks based node applications and systems. These tools include a software program environment for developing and debugging applications at multiple nodes. There are two emulators used to simulate a node with its application in the development phase. A network manager is used to install and configure these nodes, and a protocol analyzer is used to record and analyze the network message traffic. This will provide information on the overall system performance. The developer can then adjust the network configuration to obtain the

maximum network performance and to debug any errors (Echelon, LonWorks Products Manual).

6. Network Services Tools

LonWorks provides tools for installation, configuration changes, diagnostics, repair, and monitoring the entire system. It is the LonWorks Network Services (LNS) which has the software components for developing system-level applications. This LonWorks Network Services architecture ensures the compatibility and interoperability among all the nodes which are developed by different vendors. The LNS provides the tools to ensure the nodes from different developers will work together. As long as they meet the LonWorks interoperable specification for developing their node, each developer does not need to worry about the details of any other developer's design and loosing synchronization with the network's configuration. Therefore, system installation can be worked in parallel. System repair can be done at any possible problem spot in the network by using this LNS tool. LNS can save time for system installation and repair, reduce cost and increase the overall productivity (Echelon, LonWorks Products Manual).

E. SUMMARY

This chapter introduced the background and technology of control networks as provided by LonWorks. This technology provides a reliable and efficient networked control system for many different industrial applications. These applications are used in a wide range of control and propulsion systems. This technology is an open, decentralized networked control system and is different from the proprietary control and centralized systems currently used in most industrial applications. Control network technology can provide reliable, interoperable and robust networked control systems for a variety of applications.

V. PROTOCOL AND NETWORK STRUCTURE OF LONWORKS

A. INTRODUCTION

Echelon LonWorks uses LONTALK as its networking protocol. This protocol follows the International Organization for Standardization Open Systems Interconnects (ISO/OSI) reference model. The addressing hierarchy of LonTalk has three levels. They are domain, subnet, and node. The data communication modes are single-ended, differential, and special-purpose. LonWorks uses Standard Network Variable Types (SNVTs) to standardize all the network communication variables. The programming model of LonWorks is Neuron C, a C language based on ANSI C. This chapter summarizes the LonTalk protocol and network communication methods for LonWorks. The complete information is stated in Echelon Neuron chip data book listed in the references.

B. LONTALK PROTOCOL

1. LonTalk Protocol

The LonTalk protocol uses all three CPUs of the Neuron chip to implement a complete networking protocol. This protocol design follows the ISO/OSI reference model which is an open published protocol. It is a control protocol that implements all seven layers of ISO/OSI model.

This protocol is media-independent. It supports many different communication media, including wireless radio frequency, infrared, wired twisted pair, power line, and coaxial cable. It also supports multiple communication channels. A channel is one type of physical transport medium for packets. LonWorks uses a router to connect two different channels with different communication media. The LonTalk protocol supports such a network configuration to increase system flexibility and reliability.

2. ISO/OSI Model

The LonTalk implements a seven-layer protocol that is a standard developed by the International Organization for Standardization (ISO). The purpose of this development is to have an open, published general-purpose data communications

architecture. Its structuring technique is layering. The communication functions are partitioned into one of the seven layers. Each layer accomplishes its assigned tasks independently. The lower layer performs primitive functions and related services for the next higher layer. The detailed tasks performed in each layer are hidden to the one above. Therefore, the changes in one layer do not affect other layers (Stallings 97). Table 5.1 shows the seven layers in OSI model, and their relationship to the LonTalk protocol.

The first layer is the physical layer. This layer specifies the actual physical wiring that connects the network media and devices electrically. The specification of this layer consists of: types of media, range of network, number of devices per network segment, and network isolation scheme.

The second layer is the link layer. This layer specifies the rules to access the physical layer. This layer also defines the rules for framing, data encoding, CRC error checking, predictive CSMA, collision avoidance, priority access scheme, and collision detection.

The third layer is the network layer. This layer assigns the destination address for a message received from the upper layer. This destination address is part of a message packet sent to the network. This layer also provides the information for routing of messages for the network segment, and controls the bandwidth usage.

The fourth layer is the transport layer. This layer provides different levels of reliability for the message packet sent to the network. The level of reliability varies depending on the application needs. These reliability levels are: broadcast addressing, unicast addressing, multicast addressing, repeated service, acknowledged service, unacknowledged service, and authentication.

The fifth layer is the session layer. This layer initiates a request, response, or authentication message to the other nodes in the network.

The sixth layer is the presentation layer. It provides the data translation between the network variables and applications.

The seventh layer is the application layer. This layer provides the interface between the application program and the network devices.

	OSI Layer	Purpose	Services Provided	Neuron Chip's CPU
7	Application	Application compatibility	LonMark Objects, Configuration properties, SNVTs, File transfer.	Application
6	Presentation	Data interpretation	Network variables, Application messages, Foreign frame transmission, Network services.	Network
5	Session	Remote actions	Request/Response, Authentication, Network services.	Network
4	Transport	End-to-end reliability	Acknowledged and unack message, Common ordering, Duplicate detection.	Network
3	Network	Destination addressing	Unicast and multicast addressing, Routing information	Network
2	Link	Media access and framing	Framing, Data encoding, CRC error checking, Predictive CSMA, Collision avoidance, Priority, Collision detection.	MAC
1	Physical	Electrical interconnect	Media-specific interfaces and modulation schemes	MAC, XCVR

Table 5.1 Seven Layers of ISO/OSI Model (Echelon, Neuron Chip Data Book)

3. LonTalk Addressing Schemes

The addressing hierarchy of LonTalk consists of three levels. The top level is a domain, the middle level is a subnet, and the third level is a node. The addressing information is contained in the EEPROM on Neuron chip based node. This scheme is analogous to the addresses used by US postal service. The domain corresponds to a specific town, or city. The subnet corresponds to a road, and the node corresponds to a single house address.

The domain identifiers are used in a control network with more than one type of communication medium. Different domain identifiers can keep the applications using the

same communication medium in one group. The domain identifier is selectable and can be 0,1,3, or 6 bytes long.

The subset is the second level of addressing. There may be up to 255 subnets per domain. A subnet is a logical grouping of nodes from one or more channels. A router is used to divide nodes into different subnets in a domain. It can selectively forward packets to the desired subnet.

The node is the third level of addressing. There may be up to 127 nodes per subnet. A node is the basic component of LonWorks network. It can be a node with a device that performs some simple input/output functions. It can also be a node with a host processor and application that performs more complex tasks, such as network monitoring and analyzing.

Each Neuron chip based node contains a unique 48-bit Neuron ID. This ID is assigned during manufacturing and used as a network address during initial network installation and configuration.

The LonTalk addressing capabilities are (Echelon, Neuron chip data book):

Domains in a network:	2e48
Subnets in a domain:	255
Nodes in a subnet:	127
Nodes in a domain:	32,385

The addressing capability of LonWorks is enormous; it provides system flexibility and expandability.

4. LonTalk Messaging Services

There are four basic types of messages, Acknowledged (ACKD), Request/Response (REQUEST), Repeated (UNACKD_RPT), and Unacknowledged (UNACKD). The first two types require acknowledged messages between end-to-end nodes that provide better reliability services. The last two services do not require acknowledged messages from the receiving nodes. They conserve bandwidth and provide faster services with less reliability.

When a sending node sends out a message to a node or group of nodes, it expects acknowledged messages from each receiving node. This type of message is Acknowledged (ACKD). The sending node will retry to send out the same message if it

doesn't receive the acknowledged messages in a preset time frame. The network CPU of a Neuron node is responsible for generating the messages and acknowledgements without intervention of the application CPU. Each message has its own transaction ID. The Neuron nodes use these IDs to keep track of all the network messages and acknowledgements. This can prevent any duplicated messages received by individual nodes.

The Request/Response (REQUEST) service is similar to the ACKD. The only difference is that the incoming message is processed by the application CPU of the receiving node before a response is generated. The response message may include data or other useful information to the sending node. This service is designed primarily for client/server application.

The third type of service is Repeated (UNACKD_RPT) without acknowledgements. A sending node sends out a message to a node or group of nodes in preset multiple times. It does not require acknowledged messages from the receiving nodes. This service is designed primarily for multicasting its messages to large groups of nodes. It conserves the network bandwidth without overloading the network and provides a better response time.

The last type of service is Unacknowledged (UNACKD). A sending node only sends out its message once to the receiving nodes. It does not require acknowledged messages from the receiving nodes. This service provides the highest performance and transmission rate in the network traffic, and conserves the network bandwidth. However, it is a very unreliable service and the application must not be sensitive to the loss of a message.

C. DATA COMMUNICATION MODES

There are three modes of operation in the LonWorks data communication. Those modes are single-ended, differential, and special-purpose. Differential Manchester coding is used by the single-ended and differential modes. This encoding scheme is polarity insensitive, and thus reversal of polarity in the communication link will not affect data reception. It is a widely used coding scheme and reliable format for transmitting

data over various media. Table 5.2 provides the data rate over the LonWorks network with various Neuron chip input clock rates.

Network Bit Rate (Kbps)	Minimum Input Clock (MHz)	Maximum Input Clock (MHz)
1250	10.0	10.0
625	5.0	10.0
312.5	2.5	10.0
156.3	1.25	10.0
78.1	0.625	10.0
39.1	0.625	10.0
19.5	0.625	10.0
9.8	0.625	10.0
4.9	0.625	5.0
2.4	0.625	2.5
1.2	0.625	1.25
0.6	0.625	0.625

Table 5.2 Neuron Chip's Data Rate over the LonWorks Network (Echelon, Neuron Chip Data Book)

The single-ended mode and differential mode use the same data encoding scheme. In most respects, they are the same mode of operation. The difference is that the single-ended mode is used with external active transceivers interfacing to media such as free topology twisted pair, radio frequency, and coaxial cable. The differential mode is used with external passive transceivers, which are able to differentially drive and sense a twisted-pair transmission line. Both modes use differential Manchester coding to encode the transmitted data and decode received data. This scheme provides a transition at the beginning of every bit period for the purpose of synchronizing the receiver clock. It is referred as clock transition.

Figure 5.1 shows the representation of zero and one for the differential Manchester coding. It is a common coding technique and provides the benefits of transmitting data over various media in a fast and reliable format. In every clock cycle, a transition occurs to represent a single bit of data. If there is no transition during the clock cycle, it represents a "one" bit. If there is a transition at the halfway point of the clock cycle, it represents a "zero" bit.

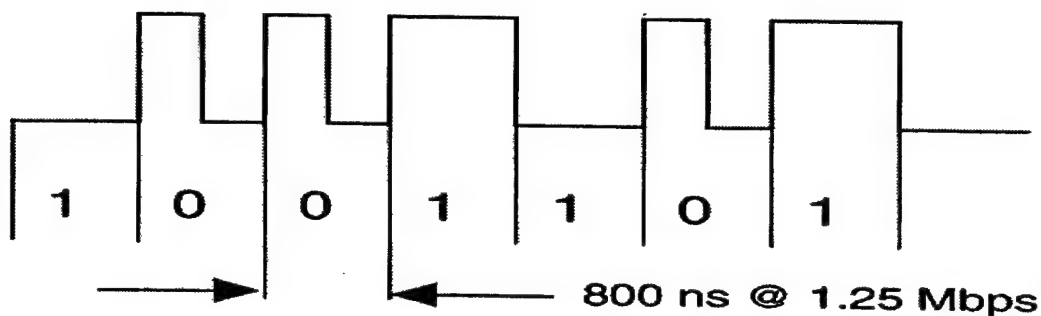


Figure 5.1 Differential Manchester Coding Scheme (Echelon, Neuron Chip Data Book)

Figure 5.2 and 5.3 show the typical transmitting data formats for the single-ended mode and the differential mode. Each message packet consists of a preamble, a data frame and address information with 16 bits of CRC, and line code violation.

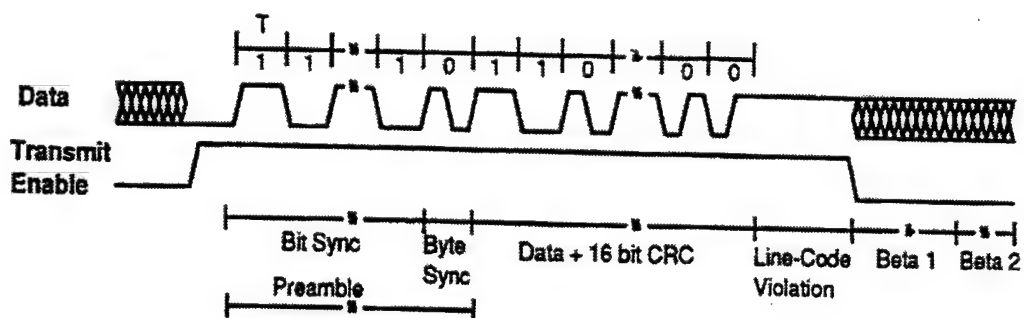


Figure 5.2 Single-ended Mode Data Format (Echelon, Neuron Chip Data Book)

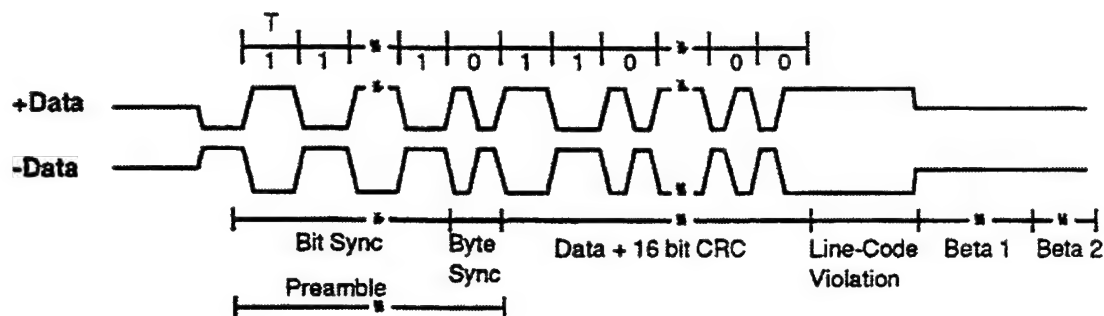


Figure 5.3 Differential Mode Data Format (Echelon, Neuron Chip Data Book)

A preamble is formed at the beginning of a packet. Its purpose is to synchronize the clock between sender and receiver. It consists of a bit-sync field and a byte-sync field. The bit-sync field is a series of differential Manchester 1's; its duration is user selectable and is at least six bits long. The byte-sync field is a single bit differential Manchester 0 that marks the end of the preamble, and the beginning of the first byte of the packet.

Followed the preamble, there is a string of data with address information. Figure 5.4 shows the typical size of a packet. Each data frame may contain the CRC as an option. The Neuron chip accepts an active-low collision detect input from the transceiver. If collision detection is enabled and the Neuron chip is signaled by one of its own communication ports, a collision has occurred. The Neuron chip acknowledges the collision and resends the message by attempting to re-access the channel at later time.

The message packet is terminated by forcing a differential Manchester line-code violation. This violation occurs when the output data is at a constant level without any transition. This level must hold long enough for the receiver to recognize an invalid code, which signals the end of the transmission. The data output can be either high or low for the duration of the line-code violation, depending on the state of the data output after transmitting the last bit.

Typical Packet Size Example

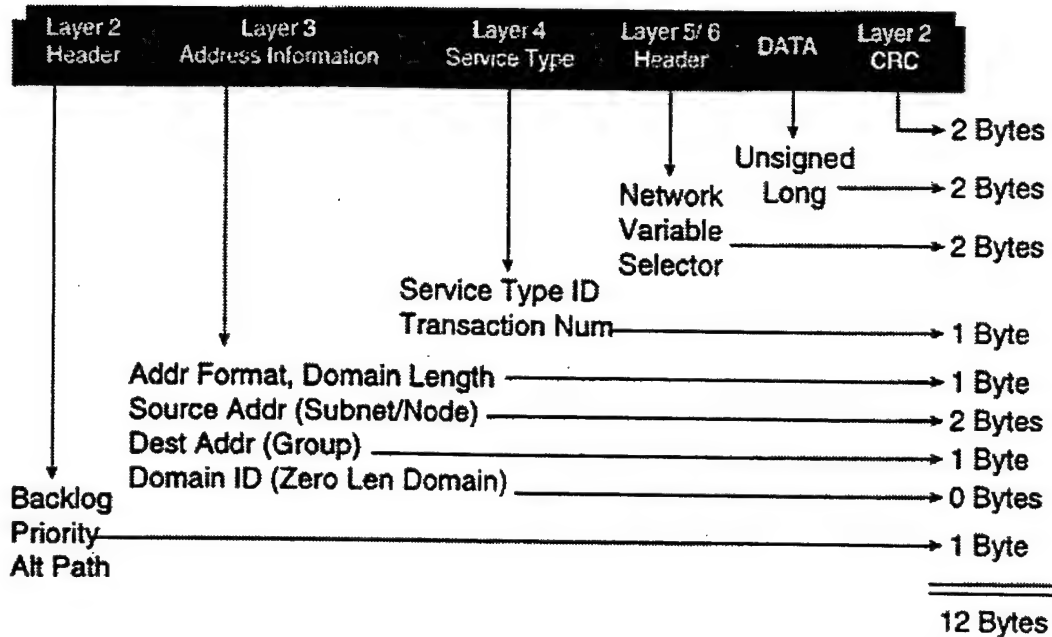


Figure 5.4 Typical Packet Size for LonWorks Network (Echelon 97)

D. LONWORKS NETWORK VARIABLES

The LonWorks application programs use the Standard Network Variable Types (SNVTs) to provide the interoperability among all the network variables of the network nodes. The application programs declare the network variables. They can be input or output network variables. An output network variable transmits its assigned value through the network to all nodes whose input network variable is binding to this output network variable. The SNVTs provide a well-defined interface for communication between nodes made by different manufacturers. A node is installed in a network and logically connected to other nodes via network variables. These input and output network variables have to match their data type. The Master SNVT List is shown in Appendix A. It includes the types of measurement, names of network variable class, its range size, and the SNVT's number classification (The SNVT Master List).

E. LONWORKS PROGRAMMING MODEL

Neuron C is the programming language used to write applications for the Neuron chip. It is based on ANSI C enhanced to support input/output, event processing, message passing, and distributed data objects. Several major differences exist between Neuron C and ANSI C. Neuron C does not include a standard run-time library supporting file I/O and other features common to larger target processors, such as floating point arithmetic. However, Neuron C has a special run-time library and language syntax extensions supporting intelligent distributed control applications using Neuron chips. These extensions include software timers, network variables, explicit messages, a multitasking scheduler, EEPROM variables, and miscellaneous functions.

F. SUMMARY

The LonTalk protocol supports reliable communication in a control network system. A system compliant with LonTalk protocol standard has the benefits of insulating the developer of LonWorks-compatible products from the detailed design of reliably moving information throughout a local operating network. It also provides installers of LonWorks networks enormous flexibility in selecting and configuring nodes to meet a particular application. Finally, the predictability of network behavior under all conditions is guaranteed, which is an important criteria for reliable robot use.

VI. HARDWARE IMPLEMENTATION

A. INTRODUCTION

The NPS AUV uses the LonWorks technology to develop and simplify its networked control system onboard. This system consists of a host IBM-compatible personal computer, a LonBuilder Developer's kit, various Neuron nodes, and required input/output devices associated with the mission of the NPS AUV. The host personal computer is a 486SX-machine uses a QNX operating system to command and control all devices onboard the NPS AUV. The LonBuilder Developer's kit is a system-level development tool. It creates, compiles, and debugs the application programs for all the nodes in the system. Each Neuron node has a local processor, Neuron chip 3150, to receive, process, and transmit data via its local input/output transceiver. The NPS AUV contains various input/output devices to meet its mission requirements. The maintenance and upgrade of LonWorks network is minimal and simple since each Neuron node is developed and operated independently.

This chapter describes all the components used in this project. The discussions of these components' functions are brief. The completed detailed information is stated in their technical manuals listed in the references. The list of all hardware components and estimated cost is stated in Appendix B.

Figure 6.1 shows the overall networked control block diagram inside the NPS AUV. Figure 6.2 shows the picture of this project in close view and Figure 6.3 shows the far view.

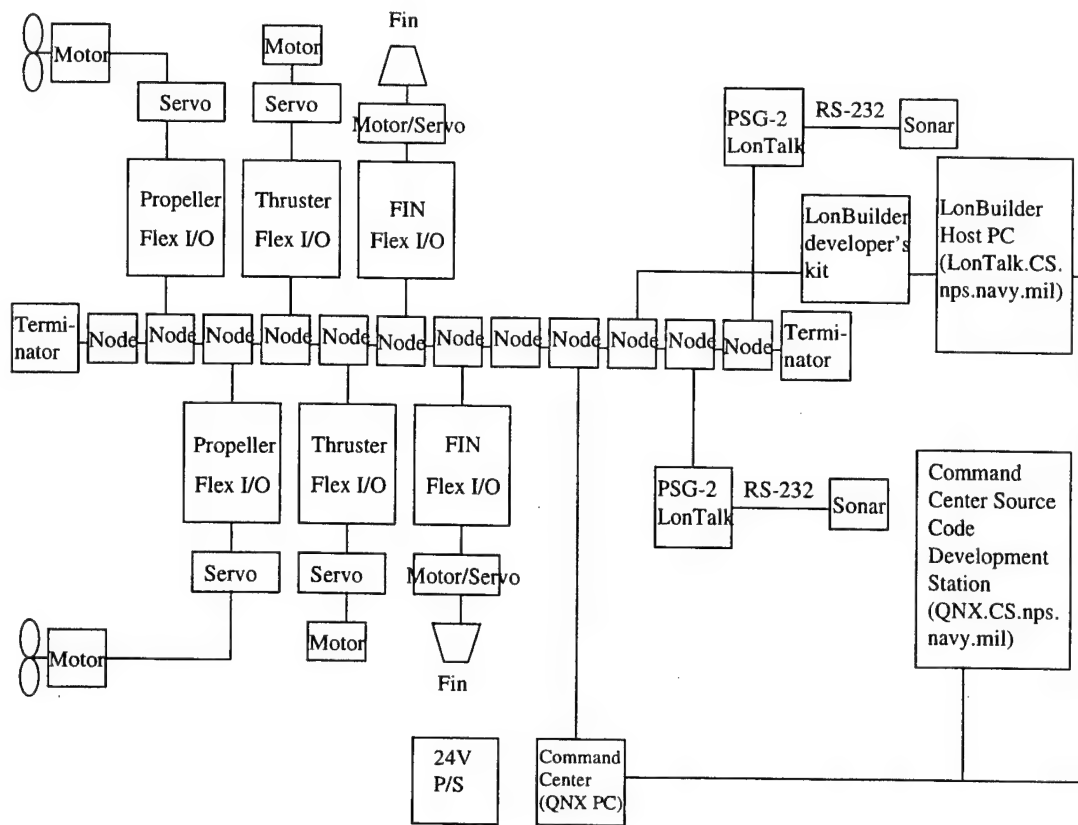


Figure 6.1 Overall Networked Control System Block Diagram in the NPS AUV

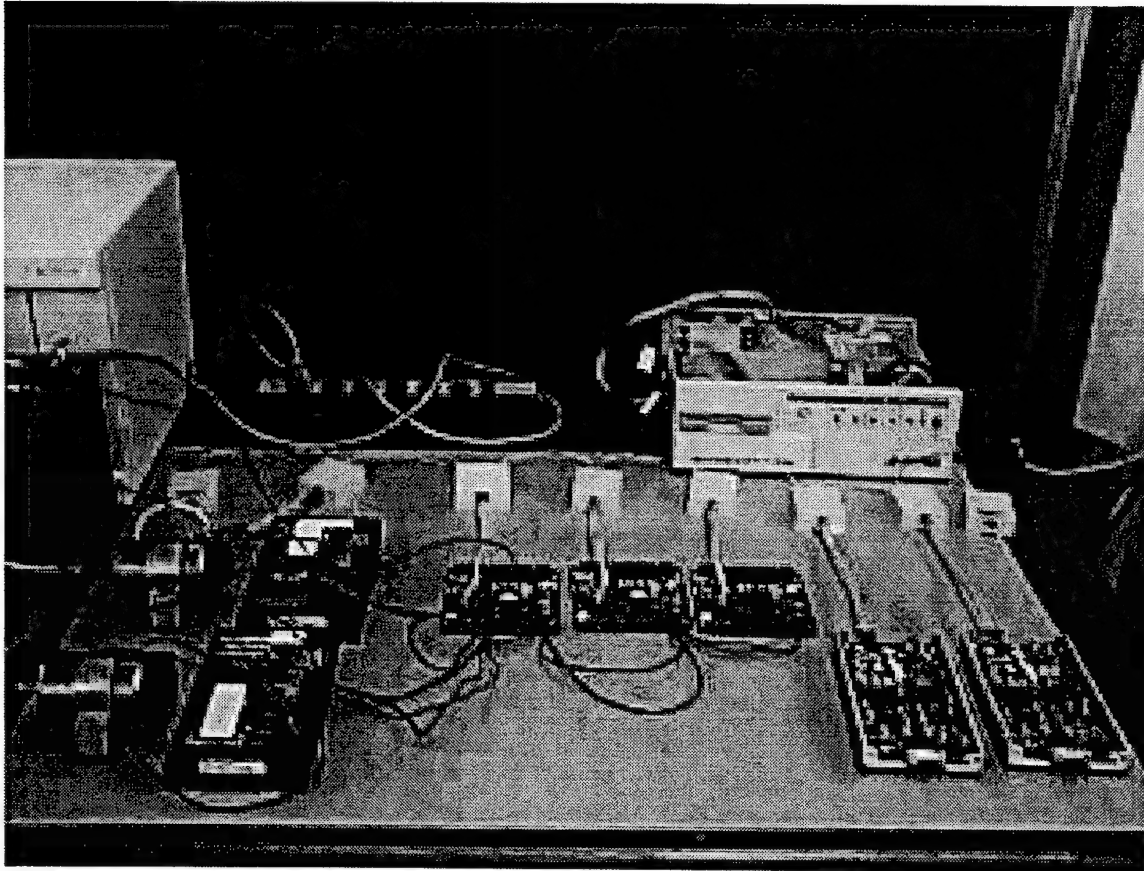


Figure 6.2 Picture of Networked Control System, Close View

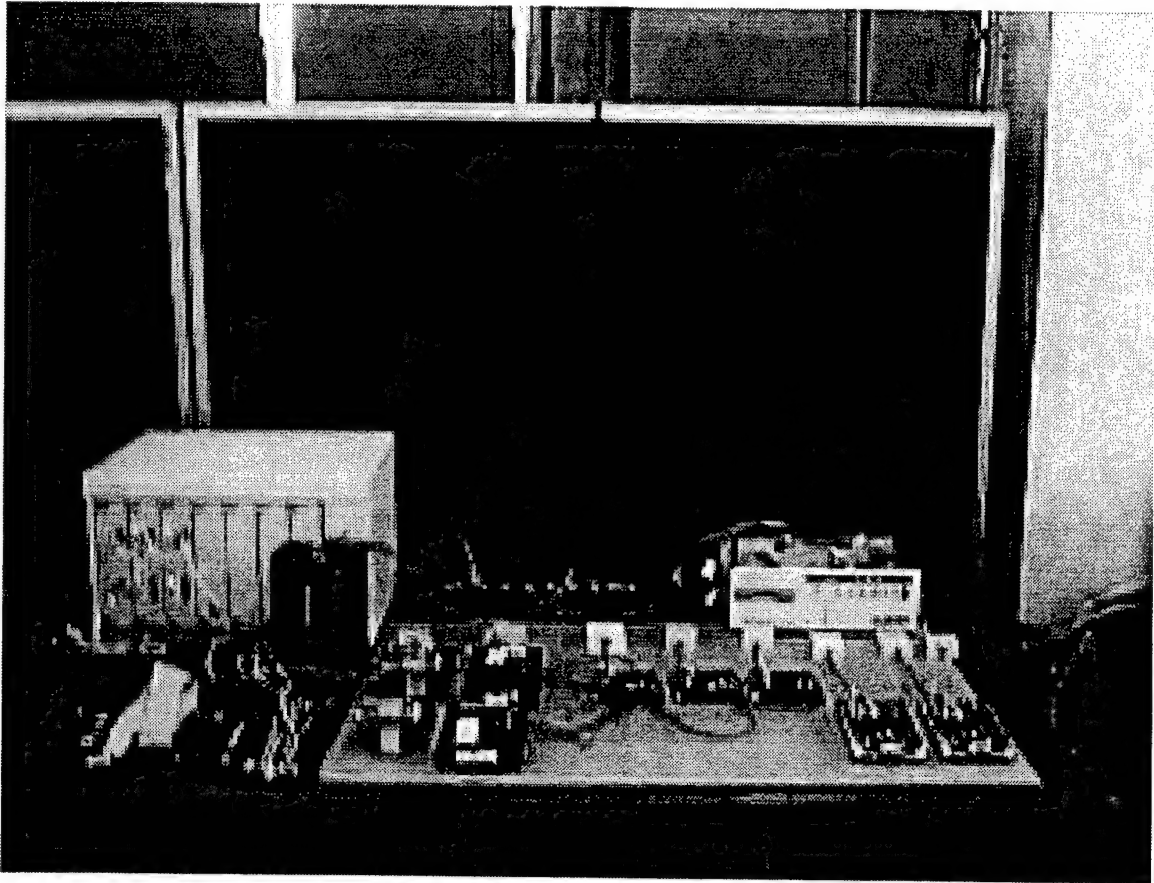


Figure 6.3 Picture of Networked Control System, Far View

B. THE LONBUILDER DEVELOPER'S KIT

The purpose of the LonBuilder developer's kit is to develop LonWorks application programs for Neuron nodes. The LonBuilder is also used to test and debug each Neuron node after the developing processes. A Neuron node is considered as a network ready device after a successful test. Connecting many Neuron nodes to a twisted pair communication media forms a LonWorks network. Once the network is formed, the network manager and protocol analyzer of the LonBuilder is used to manage the network control and analyze the efficiency and utilization of the network. Figure 6.4 shows the picture of LonBuilder developer's kit.

The LonBuilder developer's kit used in this project consists of the following components

- LonBuilder Development Station Enclosure
- LonBuilder Interface Adapter
- LonBuilder Control Processor
- LonBuilder Neuron Emulator
- LonBuilder SMX Adapter
- LonBuilder Application Interface Kit

This section summarizes the functions and usage of LonBuilder developer's kit from the Echelon LonBuilder Hardware Guide listed in the references.

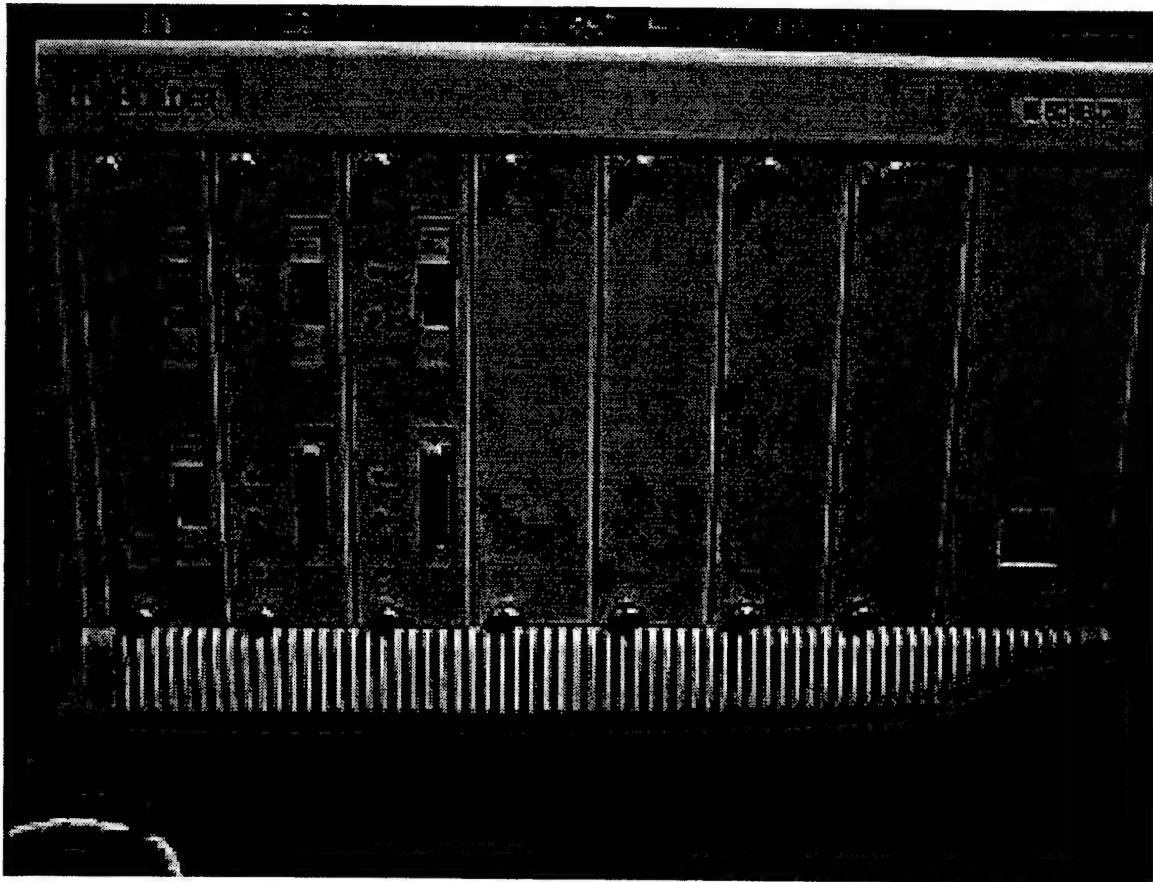


Figure 6.4 LonBuilder Developer's Kit

1. LonBuilder Development Station Enclosure

A LonBuilder enclosure hosts a control processor, a protocol analyzer, and two Neuron emulators. This development station provides the power supply, connection to the host PC, and future expandable slots. It simplifies the control network development by connecting the network manager and protocol analyzer with two Neuron emulators in one cabinet.

2. LonBuilder Interface Adapter

Figure 6.5 shows the block diagram of LonBuilder interface adapter.

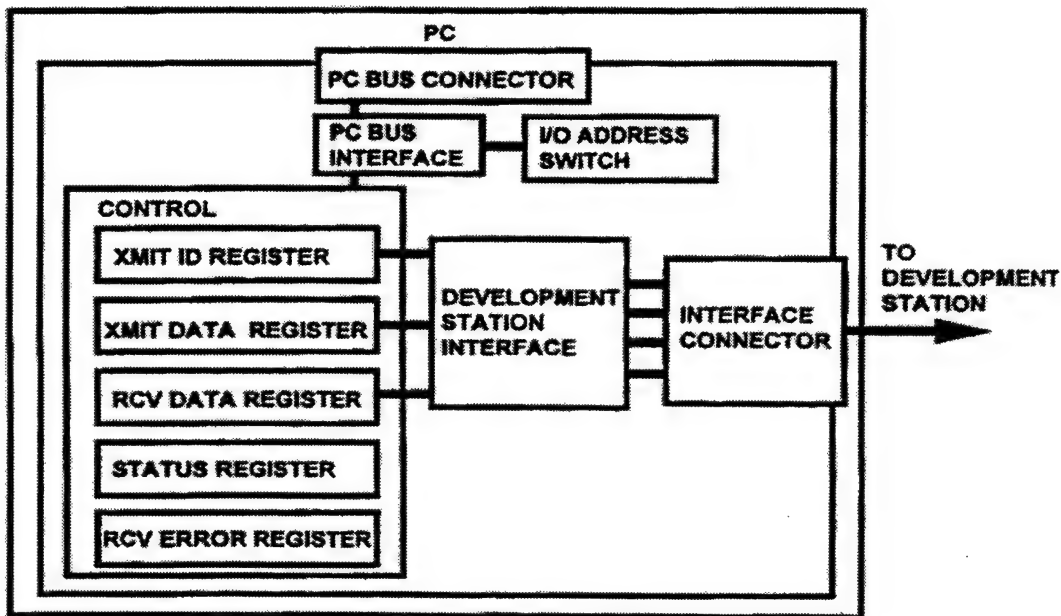


Figure 6.5 Block Diagram of LonBuilder Interface Adapter (Echelon, LonBuilder Hardware Guide)

This adapter with its LonBuilder software is installed inside the host PC. The software provides the development environment for writing, compiling, and debugging of Neuron nodes' applications. The adapter provides the connection between the host PC and the LonBuilder developer's kit. The compiled application programs are then tested in one of two emulators within the kit.

This LonBuilder interface adapter uses its eight input/output ports to communicate with the PC. Its default address assignment is set to occupy ports 310 to 317hex. The dip switch (SW1) of the interface adapter can be adjusted to assign a different I/O address within the host PC to avoid any address conflict with other devices. Table 6.1 shows a typical I/O address usage in a PC compatible computer.

I/O Address Range	Typical Use
0000-01FF	Reserved for PC motherboard hardware
0200-0207	Joystick input
0220-022F	Sound Controller
0278-027B	LPT3 Parallel Port
02F8-02FF	COM2 Serial Port
0310-0317	LonBuilder Interface Adapter
0320-0327	LonBuilder Protocol Analyzer
0330-033F	MIDI Controller
0340-0347	PC LonTalk Adapter
0350-0357	PCNSS PC Interface Card
0360-036B	PC Network
0378-037B	LPT2 Parallel Port
0388-038F	Sound Controller
03B4-03BA	Video Subsystem
03BC-03BF	LPT1 Parallel Port
03C0-03DA	Video Subsystem and DAC
03F0-03F7	Floppy Disk Controller
03F8-03FF	COM1 Serial Port

Table 6.1 Typical Input/Output Address Assignment in a PC (Echelon, LonBuilder Hardware Guide)

3. LonBuilder Control Processor

The control processor provides control and network management between the interface adapter in the host PC and the processor board within the development station. It handles the commands and data communications between the interface adapter and the processor boards for the Neuron nodes. The control processor board contains a Network Manager node and a Protocol Analyzer node. The purpose of the network manager is to define, configure, load and control LonWorks nodes and the network's operations. The purpose of the Protocol Analyzer is to monitor, collect and display network traffic and network performance statistics.

Figure 6.6 shows the picture of this control processor board and Figure 6.7 shows the control processor block diagram.

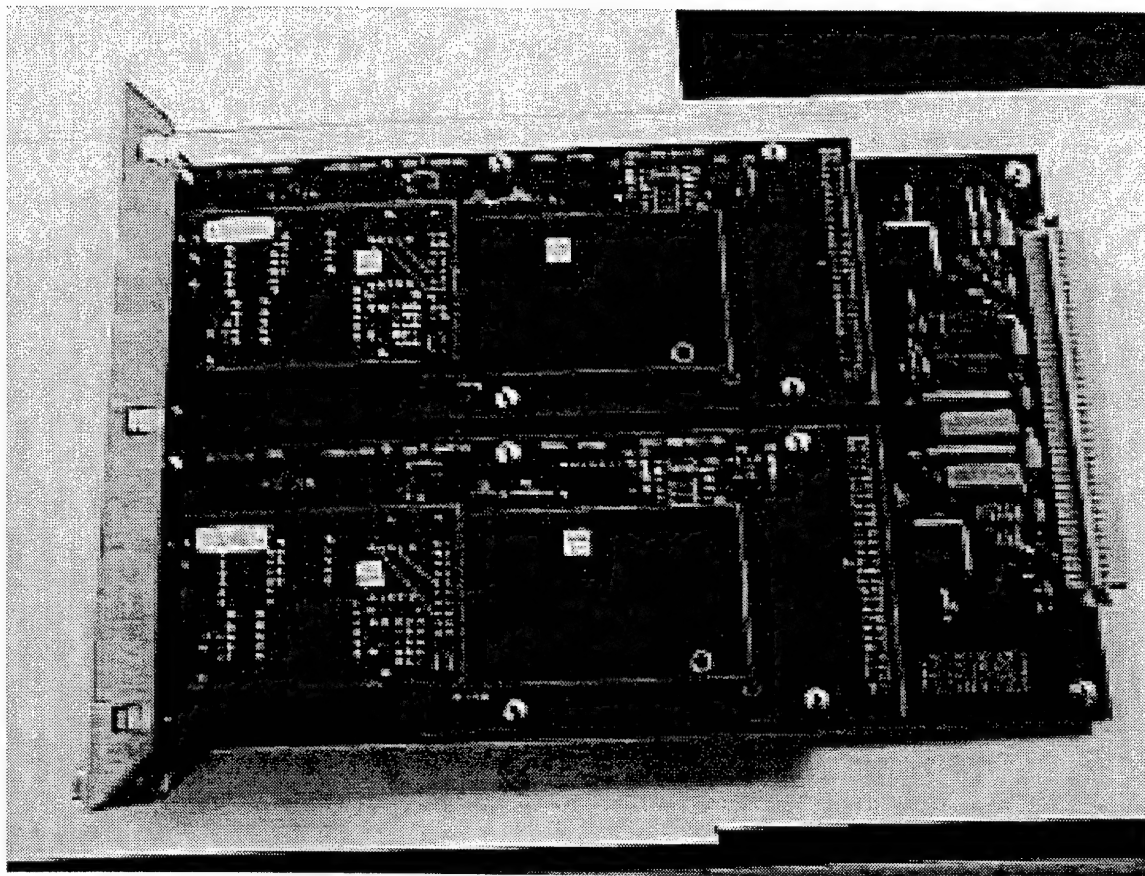


Figure 6.6 Picture of Control Processor Board

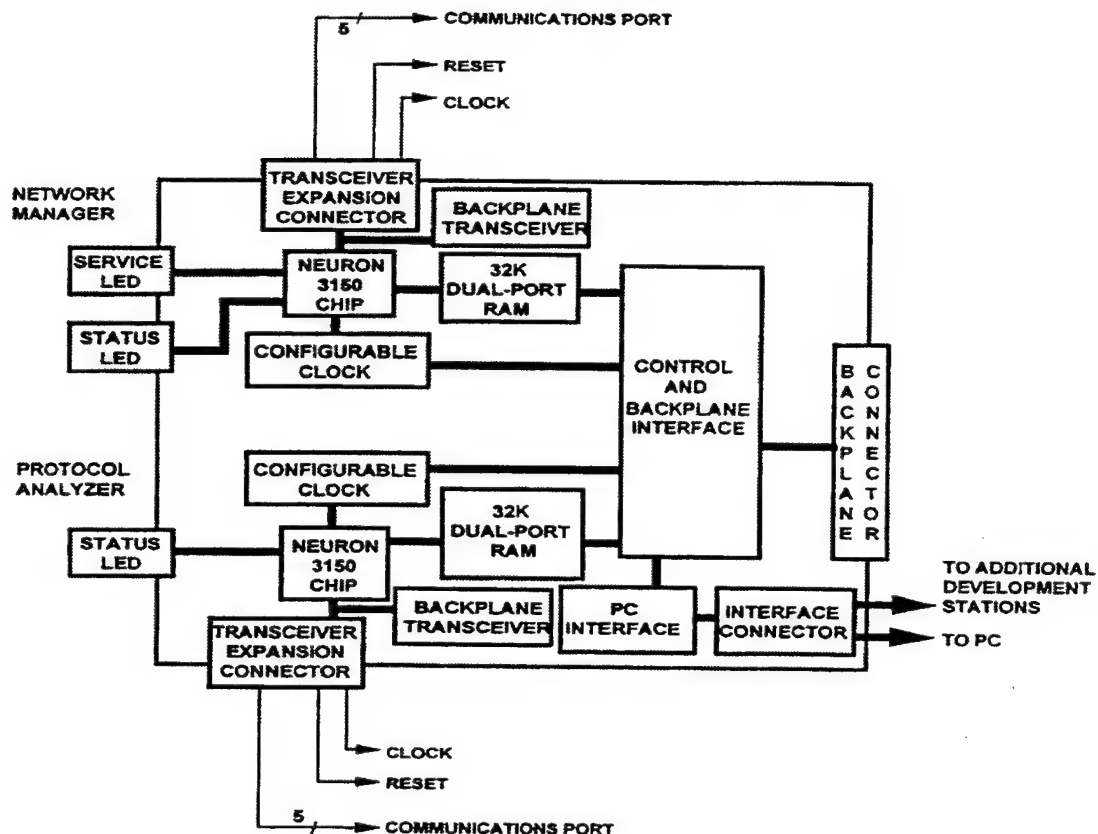


Figure 6.7 Block Diagram of LonBuilder Control Processor (Echelon, LonBuilder Hardware Guide)

4. LonBuilder Neuron Emulator

The purpose of LonBuilder Neuron Emulator is to test and debug the node application programs and support hardware prototyping. Figure 6.8 shows the picture of the Neuron emulator and Figure 6.9 shows the Neuron emulator block diagram.

A Neuron emulator contains a Neuron 3150 Chip, 64Kbytes of code and data RAM, and 64Kbytes of control RAM to act a single Neuron node. It uses a LonBuilder SMX adapter and transceiver to transmit data over the network. The emulator provides hardware support for application loading, source-level breakpoints, single-stepping, reset, start, stop, and memory read/write protection for each Neuron node's initial development.

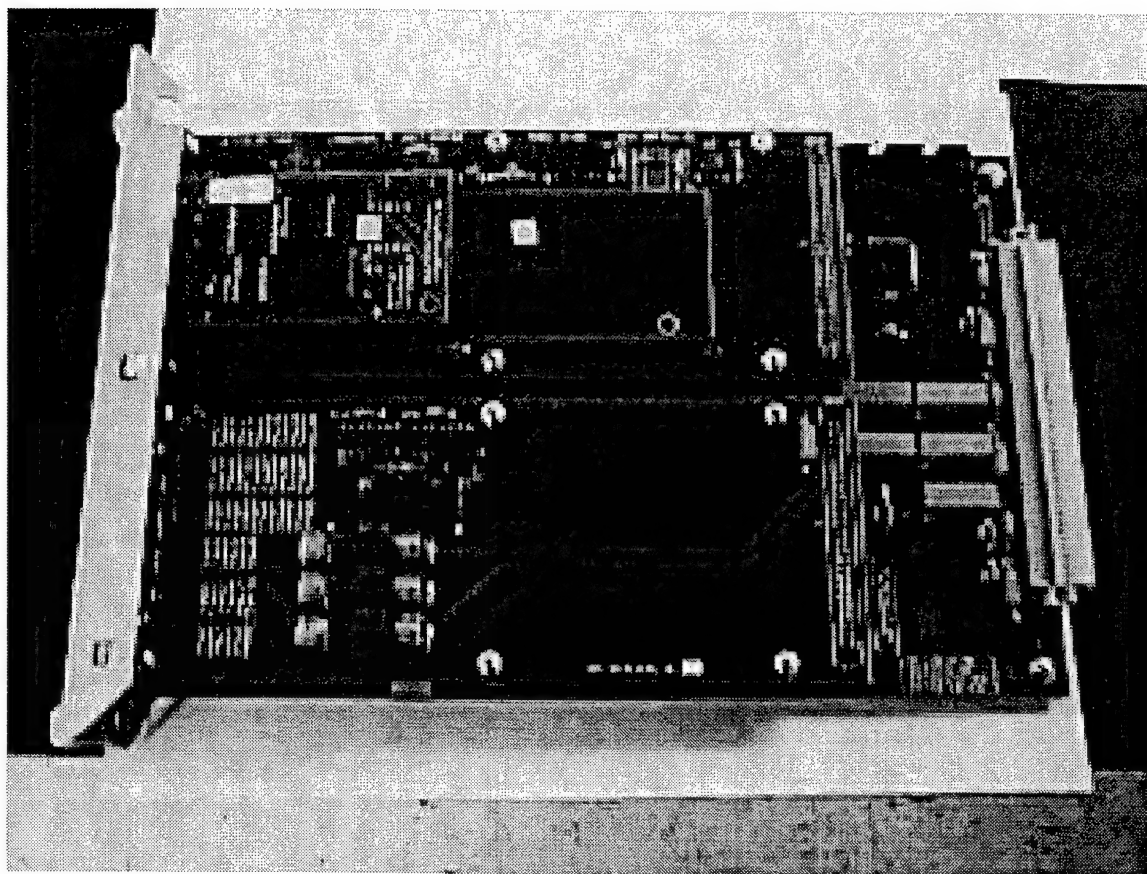


Figure 6.8 Picture of LonBuilder Neuron Emulator

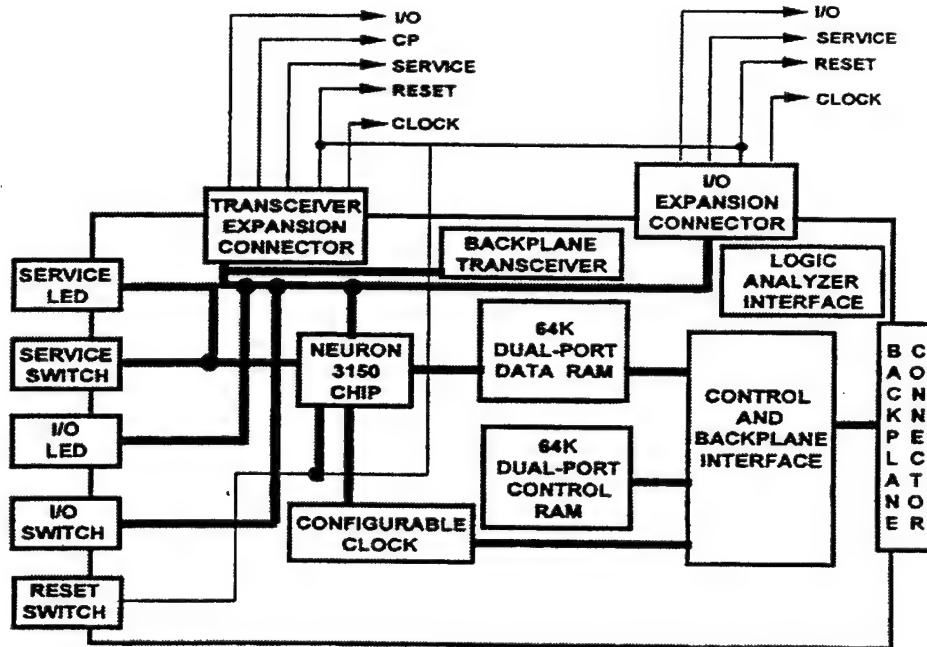


Figure 6.9 Block Diagram of LonBuilder Emulator (Echelon, LonBuilder Hardware Guide)

5. LonBuilder SMX Adapter

This adapter is an interface board that provides the connection between SMX-compatible transceivers and LonBuilder processor boards (control processor boards and emulator boards). It provides a modular, flexible solution for interfacing LonBuilder processor boards with a wide variety of network media, such as twisted pair or power lines.

6. LonBuilder Application Interface Kit

The LonBuilder application interface kit contains one application interface board, one application cable, one application interface adapter, and one module application interface. Figure 6.10 shows a picture of this kit.

The LonBuilder application interface board provides a means of connecting the Neuron chip on a Neuron emulator directly into a user's target application hardware. This kit provides a developer the ability to design and debug many different custom nodes for the LonWorks networked control system.

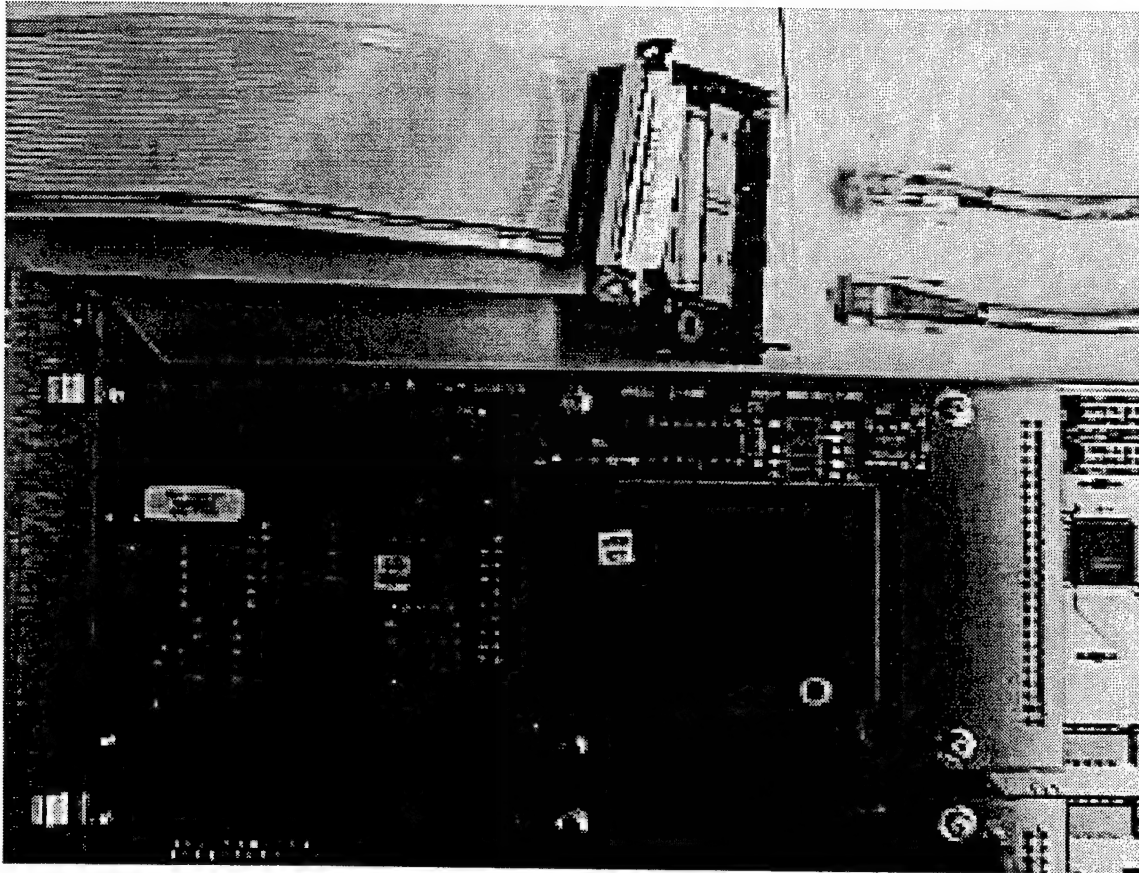


Figure 6.10 Picture of LonBuilder Application Kit

C. LONWORKS NEURON NODES

A typical LonWorks Neuron node consists of

- A local processor, Neuron chip: it is a microcontroller that performs local data processing and application functions, implements LonTalk protocol, and accesses the network media.
- An input/output interface: it provides the interface between a Neuron node and its input/output external devices, such as sensors, actuator, or controller.
- A transceiver: it provides the connection between the Neuron node and the network communication media.
- A firmware and I/O application library: it provides the information of protocol, scheduler, and I/O application library for each Neuron node.

- ROM, RAM, and EEPEOM: it stores the application code, firmware data, system images, network images, and LonWorks network variables.

This project uses two different types of Neuron nodes for its initial bench testing which uses LonWorks as its networked control system for the *Phoenix*. The first type of Neuron node has the capability to control analog devices, such as motor controllers and pulse width modulators. The second type of Neuron node has the capability to control serial devices, such as sonar.

The first type of Neuron node used in this project is a Flexible I/O control module made by the Intelligent Technologies Corporation (IEC). Figure 6.11 shows the picture of this node and Figure 6.12 shows its block diagram.

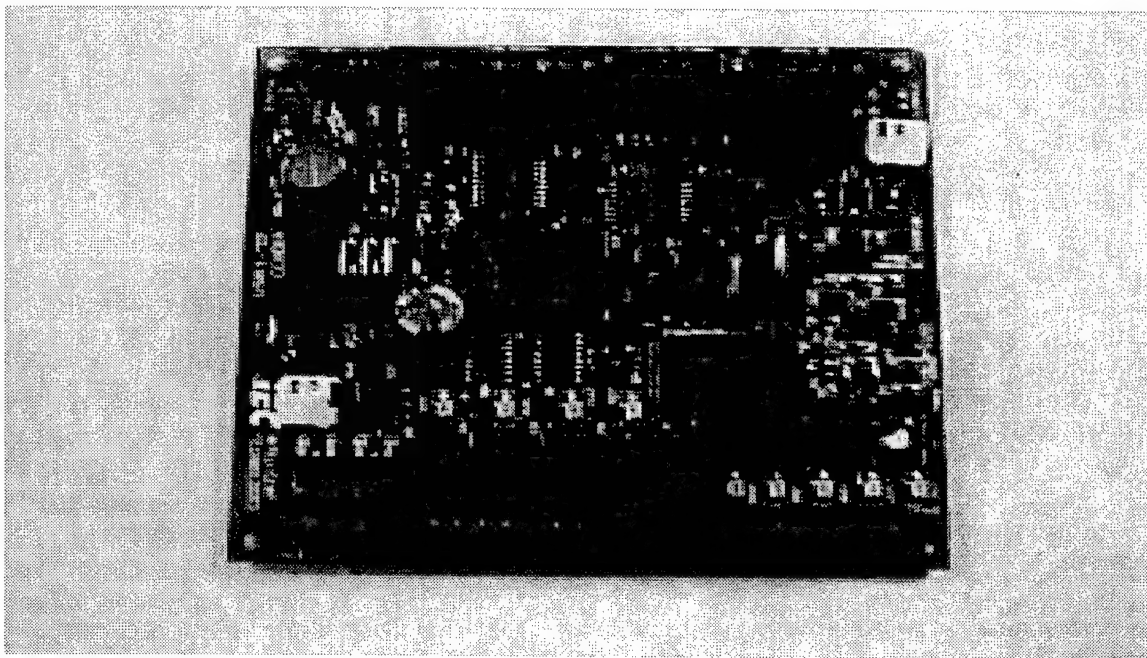


Figure 6.11 Picture of IEC Flexible I/O Node

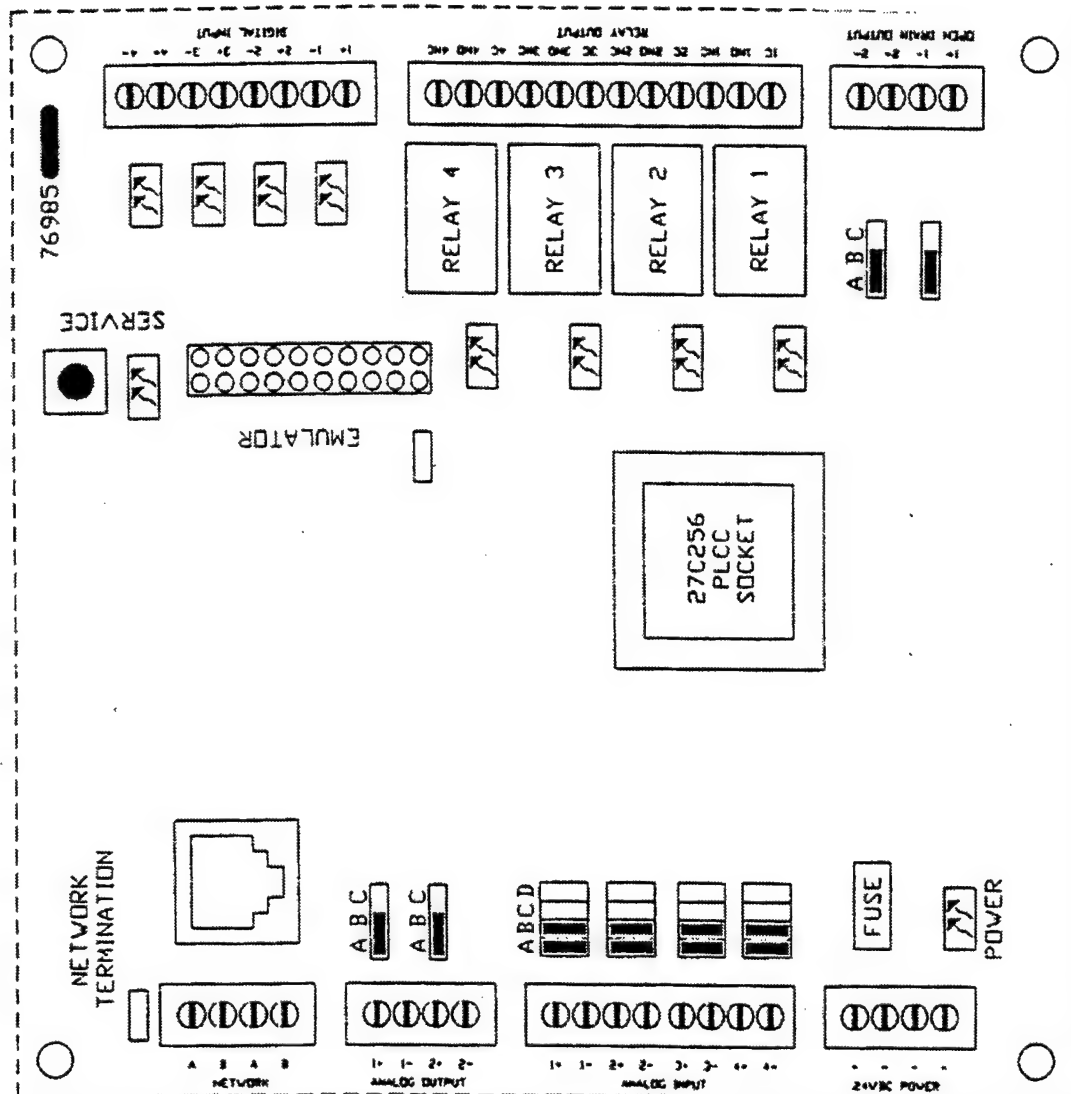


Figure 6.12 Block Diagram of IEC Flexible I/O Node (IEC, Flexible IO Users Manual)

This node is designed to be a general purpose LonWorks node. It uses a Motorola MC143150 Neuron chip as its local processor running at 5 MHz clock speed. It is a node for implementing, monitoring and controlling functions in a LonWorks control network. Its application code is written, compiled and debugged by using the LonBuilder developer's kit. The detail of this control module's software development is discussed in Chapter VII, software methodology. Once the application code is successfully developed, it is written into an off-chip EEPROM for the Flexible I/O node. An EMUP chip burner is used to load the application code into the EEPROM. Figure 6.13 shows this chip burner. The EEPROM with its application code is then inserted into the 27C256 PLCC socket onboard the Flexible I/O control module. A Flexible I/O node with its EEPROM loaded with application code is a network ready Neuron node.



Figure 6.13 Picture of EMUP Burner

The Flexible I/O node consists of two sets of analog output, four sets of analog input, four sets of digital input, four sets of relay output, two sets of open drain output, and two sets of network connections. There are screw terminals, LED indicators, a network SERVICE switch and a RJ-45 network connector which are used for network installation and diagnostics (IEC, Flexible I/O users manual).

This project uses the analog output of a Flexible I/O node to control the servo amplifiers that connect to many external devices. The servo amplifiers receive signals from the control network to control the propeller and thruster motors onboard the NPS AUV. The analog output signal is configured for 0-10 Volts DC (VDC) to control various motor speeds.

The open drain output of this node controls the pulse width modulation (PWM) for control of the fin motors on board the NPS AUV. It supports 1 Amp, 60 VDC with a repetition rate of 19.531 KHz (IEC, Flexible I/O users manual).

The second type of Neuron node is a Serial to LonTalk Adapter (SLTA). This project uses SLTAs to connect all the sonars that have EIA-232 serial interfaces to the communication media. Figure 6.14 shows a picture of SLTA.

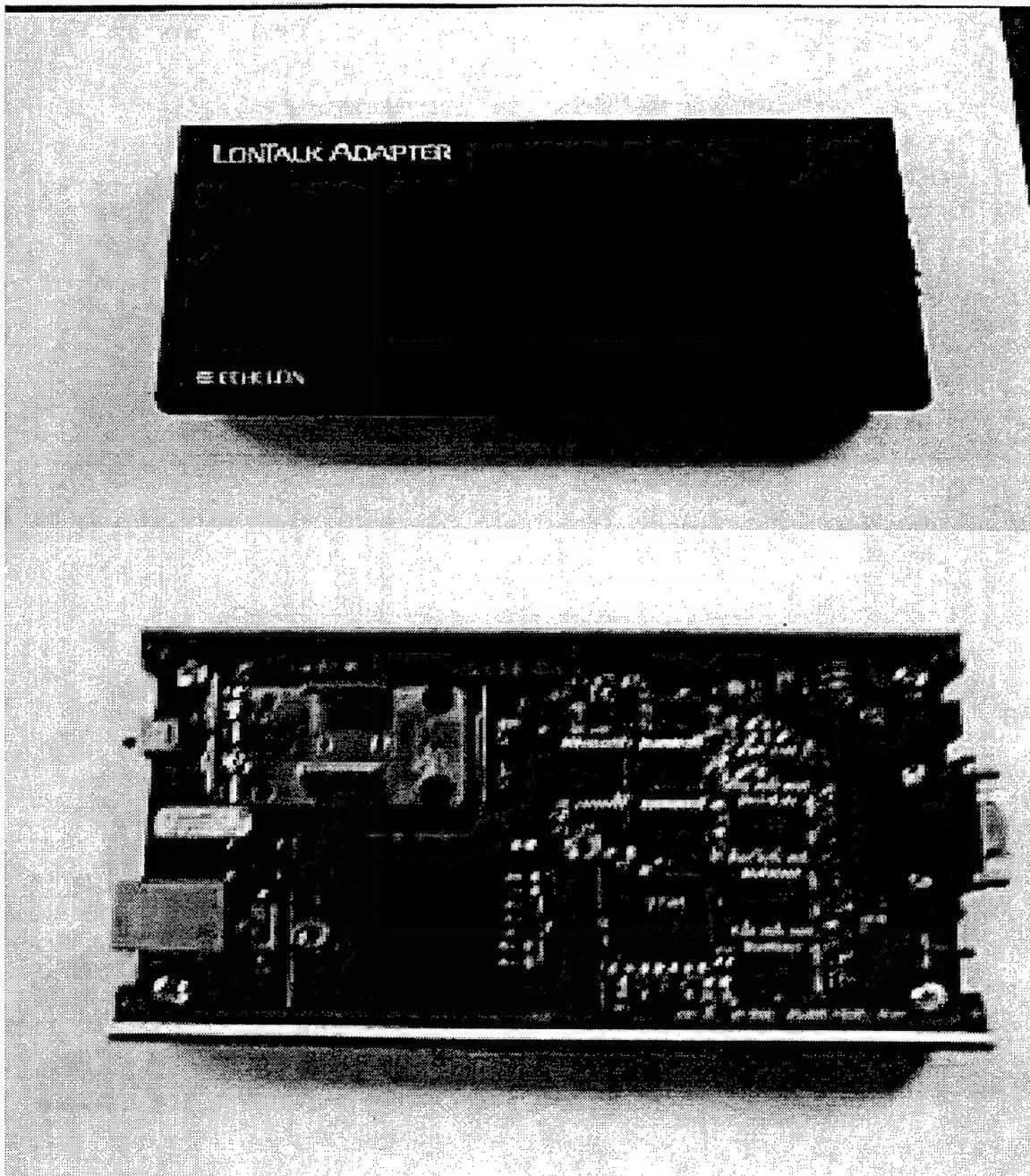


Figure 6.14 Picture of Serial to LonTalk Adapter (SLTA/2)

D. INPUT/OUTPUT DEVICES

The input and output external devices in the NPS AUV are the following:

- Analog devices: Servo Amplifiers with motors that control propellers, thrusters and fins.
- Serial devices: ST725 Scanning sonar, ST1000 Profiling Sonar, and diver tracker for altimeter.

This project uses Advanced Motion Controls (AMC) Servo Amplifiers, model 30A8 to control analog devices. This servo amplifier is designed to drive brush-type DC motors. It is fully protected against over-voltage, over-current, over-heating and short-circuits across motor ground and power leads (AMC PWM servo amplifier operating manual). Figure 6.15 shows a picture of this servo amplifier with motors.

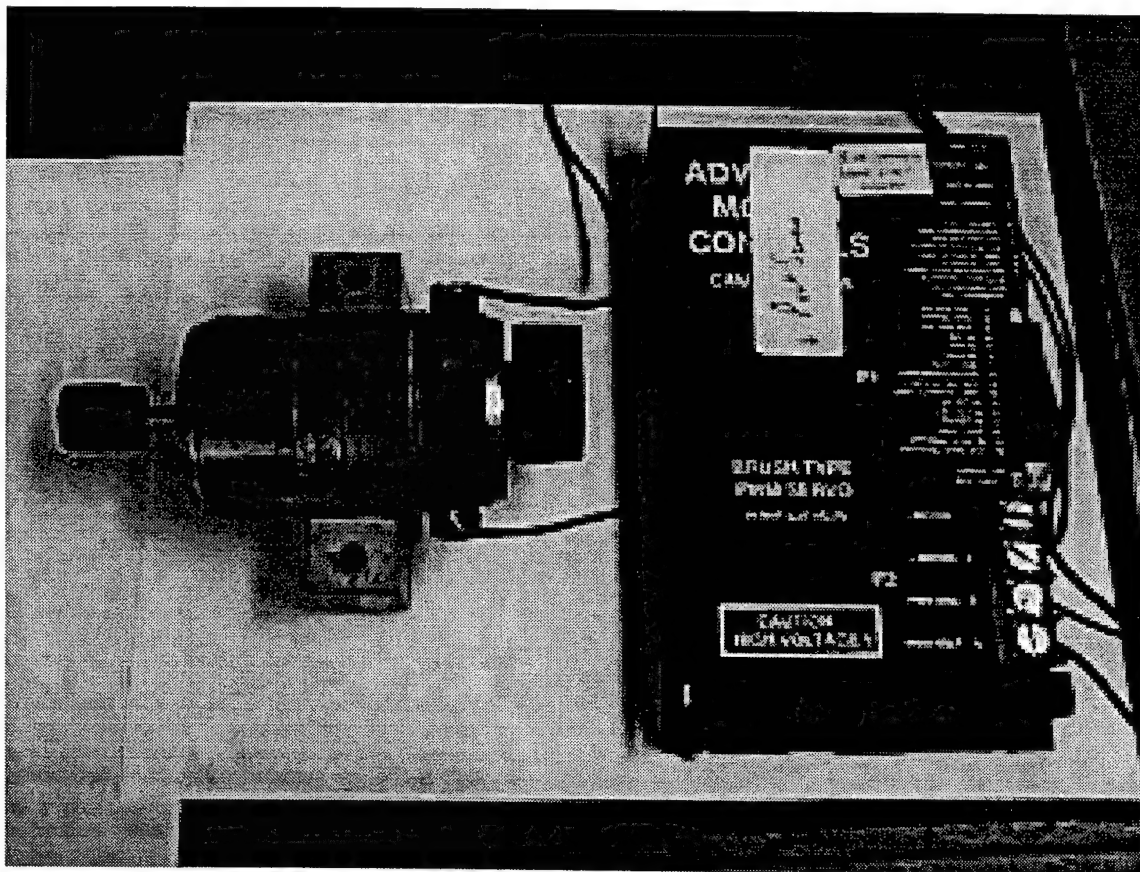


Figure 6.15 Picture of Advance Motion Controls (ACS) Servo Amplifier

This servo amplifier consists of 16 pins, 10 switches, and power supply connections for itself and the connected motors. Its internal DC-to-DC converter allows operation from a single power supply and outputs voltages of ± 10 V at 5mA for external use (AMC PWM servo amplifier operating manual). The servo amplifier connects to a Neuron node and takes an input signal from the LonWorks network to its differential pre-amp, pin 6 and 7. It uses the input signal to convert its 24 V input voltage to variable output voltage of ± 10 V. This output voltage is used to drive the motors for propellers, thrusters and fins onboard the NPS AUV.

The second type of external components are serial devices. There are two sonars and a diver tracker. The first sonar is a ST725 scanning sonar operated at 725 kHz. It is primarily used for the transit search due to its long-range search ability. The second sonar is a ST1000 profiling sonar operated at 1250 MHz. It is primarily used for sector search due to its superior range accuracy at near range. All communications with both sonars are conducted using Asynchronous RS-232 signal level at 9600 Baud, one start bit, one stop bit, and no parity bit. The network communicates to the Sonar Head via the RS-232 serial communications port COM1. In order to establish hardware handshaking, it requires some type of modem equipment (Tritech 92). This project uses the Echelon SLTA to establish this hardware handshaking between the sonars and the LonTalk network.

E. THE NPS AUV'S NETWORK CONNECTION AND IMPLEMENTATION

Once all individual devices have been developed and configured as LonWorks components, they are physically attached to the LonWorks control network system in a bus topology. Figure 6.3 shows the picture of this network. This system uses a Level IV, 22 AWG (0.65mm) twisted pair cable as the network's primary bus. This type of cable can sustain 1.25 Mbps data communication with a TPT/XF-1250 control module and transceiver on each Neuron node.

This project uses three IEC Flexible I/O nodes to control all of the analog devices. One node is connected with two servo amplifiers for the propeller motors. Another node is connected with two servo amplifiers for the thruster motors. The last node is

connected with two more servo amplifiers for the fin motors. It also uses one SLTA node to connect one sonar serial device. Table 7.4 of Chapter VII lists the name of each Neuron node and their network variables. They are created during the development of application program source code.

The Neuron nodes are connected to the LonWorks control network one at a time. Once a new Neuron node is connected to the network and powered up, its service pin is pushed to identify itself by sending its 48-bit unique Neuron ID to the LonBuilder's database. The LonBuilder accepts this incoming new node and issues a queue command to extract all of its the network variables. These network variables are stored inside the LonBuilder's database with its unique Neuron node.

When all the Neuron nodes have been connected to the network and all network variables have been extracted and stored in the database, a binding process is carried out by the LonBuilder. A binding process provides the interoperable communication between Neuron nodes by using their network variables. The application program of each Neuron node determines the types, directions, units, and ranges for its network variables.

The propeller node uses two of its network variables to control the two servo amplifiers for the propeller motors. The network variable `Nvi_raw_analog_1` and `Nvi_raw_analog_2` are of type `SNVT_count` and are scaled from 0-4096 for 0-10 VDC (Flexible IO users manual). The various output voltages are used to control the propeller turning speeds.

The thruster node is identical to the propeller node. The LonWorks network can differentiate two different nodes by recognizing each node's unique 48-bit Neuron ID even when the two nodes use the same names for their network variables.

The fin node uses `Nvi_open_drain_1` and `Nvi_open_drain_2` to control the fin motors. The open drain MOSFET transistor outputs support the pulse width modulation (PWM). This node controls two fin motors, which are variable speed reversing DC motors for ± 10 VDC. The scaling for these network variables are 0-255 for 0 to full modulation. The repetition rate is 19.531 KHz and the pulse width frequency changes at the end of the current cycle (IEC, Flexible IO users manual).

The sonar node uses Char_in and Char_out as its input and output network variables. This project uses a portable PC to simulate a serial device. It connects to a SLTA via RS-232. The SLTA is connected to the network media via RJ-45 network connector. The network variable browser of the network manager can be used to send out a string of characters in a packet format. This packet message passes through the SLTA and is received by the portable PC. This string of characters is then displayed on the screen of the PC. Conversely, the portable PC can also send out a string of characters that pass through the network and back to the LonBuilder, which can be viewed by the network variable browser.

F. LONWORKS NETWORK'S UPGRADE, MAINTENANCE AND REPAIR

Upgrading a LonWorks networked control system by adding or removing Neuron nodes is an easy task. Adding a new Neuron node to the network is the same as the tasks described in the above section. Removing a Neuron node is as simple as unplugging the node's RJ-45 connector from the network. The LonBuilder's database can still keep the Neuron ID of the removed node and other related information. It will not affect any further network operation. In brief, adding or removing a Neuron node does not require any network reconfiguration.

The maintenance of LonWorks network system is rendered by the network manager and protocol analyzer in the LonBuilder. These two nodes are used to manage, monitor, and log the LonWorks network activities. The network manager uses its network statistics to display the network's performance and utilization. Its packet log is used to collect copies of actual message packets on the network. It states the time, type, and the addresses of the sender and receiver for each message packet.

The network variable browser of the protocol analyzer can display and monitor the value of each network variable. When a device does not respond to its input command, the network variable browser can display the values of the network variable both in sending and receiving nodes. Therefore, the problem area can be easily located and isolated.

G. SUMMARY

This chapter summarized all the hardware components used in this project. The components used in this workbench testing are only a representative portion of actual AUV components. However, this project includes all types of external devices used in the NPS AUV and demonstrates a working model by using LonWorks technology. The NPS AUV uses two types of Neuron nodes for its control network system. One is for A/D D/A devices and the other one is for serial devices. Each Neuron node is developed and installed into the network by LonBuilder. Since each Neuron node is developed and operated independently, adding or removing a node is a simple task performed without reconfiguring the entire network system. The LonWorks system in the NPS AUV simplifies the maintenance and makes troubleshooting easier.

VII. SOFTWARE METHODOLOGY

A. INTRODUCTION

The software of the LonBuilder development kit provides the basic tool to develop the application programs in each Neuron Chip based node in the network. This software consists of project manager, network manager, protocol analyzer, program editor, Neuron C compiler, and debugger. Figure 7.1 shows all components of this development kit. These programs are combined together in the LonBuilder Integrated Development Environment (IDE) which support the application programming of each Neuron node. Section B of this chapter summarizes the basic development process of a LonWorks application program. The completed description is in the Echelon LonBuilder User's Guide, Neuron C Programmer's Guide and Neuron C Reference Guide, listed in the references.

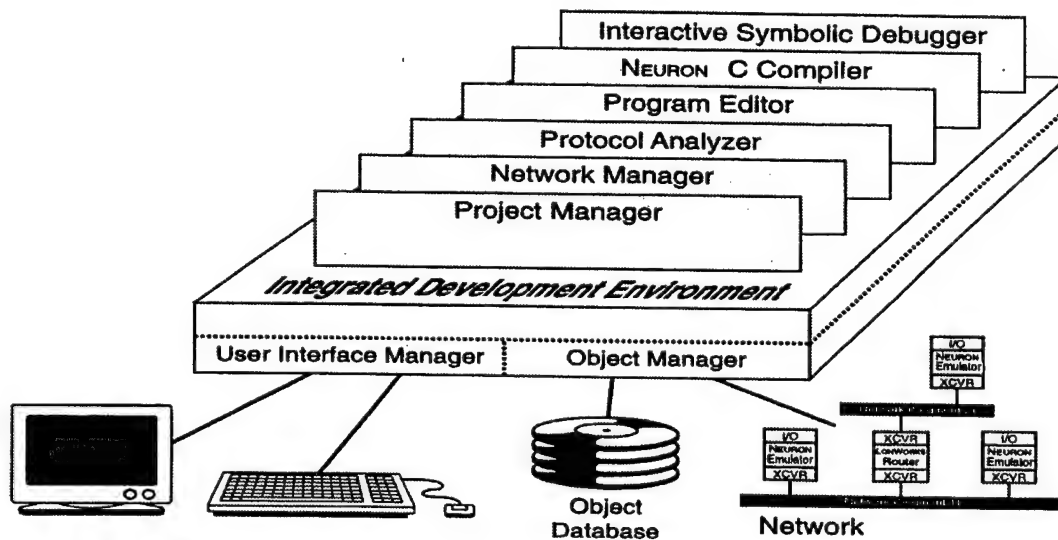


Figure 7.1 Integrated Development Environment (Echelon, LonBuilder user's guide)

This chapter also describes the development of application programs for the two types of Neuron nodes in this project. One is the A/D D/A application program, and the other one is the application program for the serial data transmission. When all the application programs have been developed and implemented into each Neuron node, they

are then connected and bonded together using their network variables. Using the network manager of the LonBuilder development kit does the binding process and forms the LonWorks network.

B. LONBUILDER DEVELOPMENT ENVIRONMENT

The workbench-testing phase for the NPS AUV consists of six nodes. Table 7.1 lists the names of nodes and external devices which connect to the nodes.

Name of Node	External devices attached to the node
Command Center	one Host PC operated in QNX operating environment
Propeller	two propellers with two motors and servos
Thruster	two thruster with two motors and servos
Rudder	two rudder with two PWM motors
Indicator	two light indicators reside on two emulators of LonBuilder
Sonar	one serial device, simulated by a portable computer

Table 7.1 Neuron Node Names and their External Devices

To develop a LonWorks application for this project involves six basic steps. These steps are general and they can be used to develop a single node with one device or many nodes with many devices in a very complex system. These steps are:

1. State the problem
2. Identify all nodes and assign their functions
3. Define the interface of the external device for each node
4. Write applications program for each device and node
5. Use LonBuilder to build, compile, debug, and test each node
6. Connect and integrate all nodes into network and test

1. State the Problem

The problems associated with current NPS AUV are slow data processing rate, complex network architecture, and lack of real-time response. The communication and networked system onboard the AUV is complicated and difficult to maintain and upgrade. The goal of this project is to simplify the current system by using the

LonWorks Technology. As stated in previous chapters, this technology uses Neuron nodes to connect external devices to form the networked control system. It is a decentralized networked control system.

2. Identify Nodes and Assign their Functions

The second step is to identify all nodes used in this project and assign their functions. This project consists of six nodes for the workbench testing of the NPS AUV. Each Neuron node is developed independently. It consists of a Neuron Chip, one or two external input/output devices, and a communication transceiver. Each of six nodes uses the Neuron Chip as a local application processor. These are called Neuron Chip hosted nodes. The communication transceiver is a TPT/XF 1250 twisted pair transceiver.

The six nodes consist of a central command node, a node to control two propeller motors, a node to control two thruster motors, a node to control two rudder motors, a node to control the serial devices' input and output, and a node to control two light indicators.

3. Define the Interface of the External Device for Each Node

During the node interface definition, each interface of the node is needed to make it visible to other nodes. The application-level LonMark objects are used to define these interfaces. The LonMark objects are those external devices connected to the Neuron nodes. They are the products that can meet the interoperable standard of the LonWorks Technology. These objects build upon Standard Network Variable Types (SNVTs) and provide an application layer interface with the Neuron nodes. The SNVTs are standard variable types that define units, ranges, and type identification. These objects use the SNVTs to provide semantic meaning about the information that has been transmitted and received by each node over the network (LonMark 96).

The LonMark objects and the SNVTs of the nodes are visible to other nodes. Each node is defined and configured base on its external device. This allows each node to be developed independently. This type of development process can maximize the interchangeability among different devices with different properties. It also can minimize the reconfiguration requirement when there is a network or application change.

Each node and input/output external device requires defining two types of network variables. One is an output network variable that is generated by the node and exported out to the network for propagation across the network. The other one is an input network variable that is received by a node from the network and used to update the node's network variable.

Table 7.2 summarizes the functions of the six nodes.

Names of Neuron Node	Functions
Command Center as Execution Level Interface	The central command of the AUV. It issues various commands to different nodes to perform the desired missions. These missions can be basic maneuvering, search and detect external objects, etc.
Propeller	This node controls the speed of two propeller motors via two servo amplifiers.
Thruster	This node controls the speed of two thruster motors via two servo amplifiers.
Rudder	This node controls the movement of two rudders.
Indicator	This node controls the two light indicators for the status of propeller motors.
Sonar	This node sends and receives serial data to and from sonar onboard the AUV.

Table 7.2 Functions of Neuron Nodes

4. Write the Application Program for Each Node

An application program defines the function of a node, its external I/O object, the SNVTs, and the tasks that the external devices to perform. The application programs for the Neuron nodes are written and compiled in Neuron C provided by the LonBuilder developer's kit. Each node uses the application program to perform its desired functions. These application programs are processed and implemented by the Neuron Chip on the node. This chip is both an application and a communication processor. The Neuron Chip consists of eleven input/output pins (IO_0 to IO_10). These pins are used to communicate with the input/output device that is connected to the node. Figure 7.2 shows the potential pin declaration.

Use of I/O Resources Potential Pin Declarations

		0	1	2	3	4	5	6	7	8	9	10
DIRECT I/O MODES	Bit Input, Bit Output											
	Byte Input, Byte Output	All Pins 0-7										
	Leveldetect Input											
	Nibble Input, Nibble Output	Any Four Adjacent Pins										
	Touch I/O											
PARALLEL I/O MODES	Muxbus I/O	Data Pins 0-7							ALS	WS	RS	
	Parallel I/O { Master/Slave A	Data Pins 0-7							CS	HW	HS	
	Slave B	Data Pins 0-7							CS	HW	AO	
	Magcard Input	Optional Timeout							C	D		
	Magtrack1 Input	Optional Timeout							C	D		
SERIAL I/O MODES	Bitshift Input, Bitshift Output	CP	CP	CP	CP	CP	CP	CP	CP	CP	CP	CP
	Neurowire I/O { Master	Optional Chip Select							C	D	D	
	Slave	Optional Timeout							C	D	D	
	I ² C I/O									C	D	
	Serial Input											
TIMER/COUNTER INPUT MODES	Serial Output											
	Wiegand Input	O1	D1	O1	D1	O1	D1	O1	D1			
	Dualslope Input	control										
	Edgehog Input											
	Infrared Input											
TIMER/COUNTER OUTPUT MODES	Ontime Input											
	Period Input											
	Pulsecount Input											
	Quadrature Input					4+6		6+7				
	Totalcount Input											
TIMER/COUNTER OUTPUT MODES	Edge divide Output											
	Frequency Output											
	Oneshot Output											
	Pulsecount Output											
	Pulsewidth Output											
	Triac Output	control										
	Triggeredcount Output	control										
		control										
		0	1	2	3	4	5	6	7	8	9	10
		High Sink			Pull Ups				Standard			

Figure 7.2 Neuron Chip I/O Pin Declarations (Echelon, Neuron C Reference Guide)

There are many different combinations can be used to configure and utilize these IO pins. This design can increase the flexibility and minimize the overall circuitry for each Neuron node.

The external devices, LonMark objects, and the network variables used to implement them were defined in step 2. The application program of each node declares the data type and direction of its network variables. Network variables are classified based on their input or output direction. A node uses the output network variables to send data to the network. The receiving node receives the data as input network variables.

Table 7.3 summarizes the node names and their network variables for this project. Figure 7.3 shows a graphical representation of each node and their network variables.

Names of Neuron Node	Names of Network Variable	Direction	Type of SNVTs
Command Central	Nvo_l_prop	Output	SNVT_count
	Nvo_r_prop	Output	SNVT_count
	Nvo_l_thruster	Output	SNVT_count
	Nvo_r_thruster	Output	SNVT_count
	Nvo_l_rudder	Output	SNVT_count
	Nvo_r_rudder	Output	SNVT_count
	Nvo_l_indication	Output	SNVT_lev_disc
	Nvo_r_indication	Output	SNVT_lev_disc
	Nvi_l_feedbk	Input	SNVT_count
	Nvi_r_feedbk	Input	SNVT_count
	Nvo_char_central	Output	SNVT_char_ascii
	Nvi_char_central	Input	SNVT_char_ascii
Propeller	Nvi_raw_analog_1	Input	SNVT_count
	Nvi_raw_analog_2	Input	SNVT_count
	Nvo_raw_analog_1	Output	SNVT_count
	Nvo_raw_analog_2	Output	SNVT_count
Thruster	Nvi_raw_analog_1	Input	SNVT_count
	Nvi_raw_analog_2	Input	SNVT_count
Rudder	Nvi_raw_analog_1	Input	SNVT_count
	Nvi_raw_analog_2	Input	SNVT_count
Indicator	Nvi_l_ledstate	Input	SNVT_lev_disc
	Nvi_r_ledstate	Input	SNVT_lev_disc
Sonar	Nvo_char_sonar	Output	SNVT_char_ascii
	Nvi_char_sonar	Input	SNVT_char_ascii

Table 7.3 Neuron Node Names and their Network Variables

Command Center

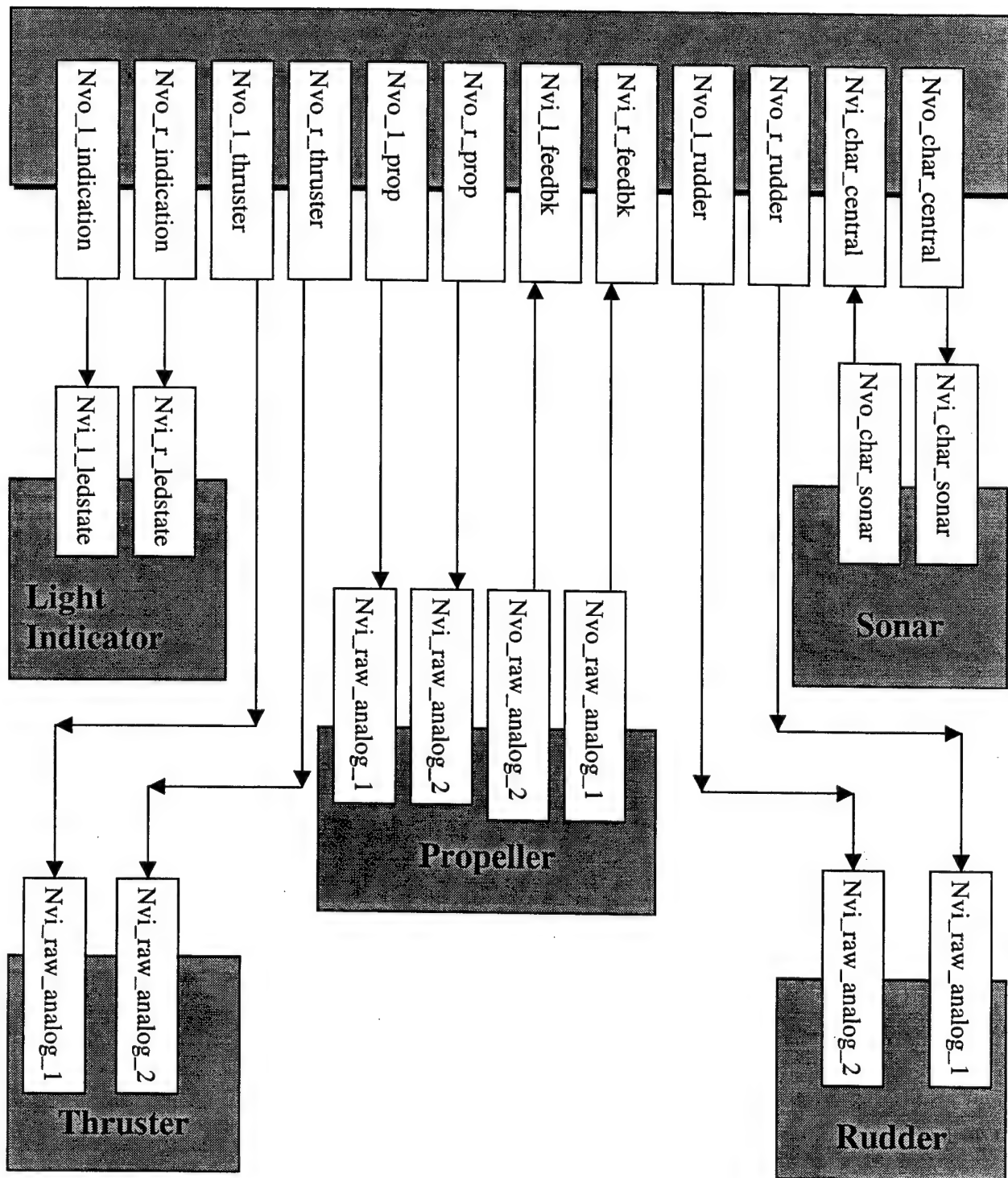


Figure 7.3 Neuron Nodes and Associated Network Variables

5. Build, Debug, and Test Individual Nodes

The LonBuilder is used to perform three tasks for each application node during the debugging step of development process. These tasks are:

- (1) Install and configure a LonBuilder Neuron Emulator in a development station
- (2) Compile and link the application code for the node, and load the application program onto the installed and configured emulator
- (3) Debug the node's application program running on the emulator using the Neuron C Debugger.

These tasks are repeated for each application node.

After a successful debugging phase, the application code is implemented in custom hardware. Custom hardware with its application code is called a custom node. This Neuron Chip based custom node contains all the components required to function as a LonWorks node.

Next, there are five tasks to build and test a custom node. These tasks are:

- (1) Design and build the custom node hardware. A typical custom node consists of a Neuron chip, an oscillator, a transceiver, off chip memory and I/O hardware. The off-chip memory contains the Neuron chip firmware.
- (2) Compile and link the application code for the node, and program the node's memory with an initial image. The initial image includes system image, application image, and network image.
- (3) Install and configure the custom node on the target communications channel.
- (4) Load the node's application program onto the installed and configured custom node.
- (5) Test the node's application program running on the custom node.

The LonBuilder network manager is used to verify that the node is functioning correctly, and its network variable browser is used to test the external interface of a node.

These tasks are repeated for each custom node. Once all the desired nodes have been installed and are functioning properly, the network can be disconnected from the LonBuilder development station and operated in a stand-alone mode.

6. Integrate Nodes into Networks and Test

A Network is built and formed from many independent Neuron nodes. During the network integration and test phase of the development process, it starts with a few nodes and gradually adds new nodes to existing, functioning systems. The network integration process involves three tasks.

- (1) Physical placement and attachment: to locate nodes in their proper places and to make any necessary attachment to the application hardware and network communication media.
- (2) Node installation: to load nodes with information that establishes the desired logical connections to other nodes.
- (3) Network test: to monitor and test communication among the nodes on the development network.

These tasks are repeated when adding new nodes to the network.

C. SOURCE CODE DEVELOPMENT FOR THE APPLICATION NODES

This project is an initial startup and workbench test to investigate the feasibility of using LonWorks for the NPS AUV networked control system. The major components of the AUV are command center, propellers, thrusters, rudders, fins and sonar. This project uses two different types of custom nodes for its external components. The first type of custom node has the ability to convert A/D and D/A signals. This project employs the IEC Flexible I/O node. The second type of custom node has a serial interface that interacts with sonar for serial data transmission. This project uses the Echelon SLTA/2 Serial LonTalk Adapter.

The IEC Flexible I/O node is designed, developed and manufactured by the Intelligent Technologies Corporation. It is a general purpose LonWorks based node. This project uses these types of nodes to control the motors of propellers, thrusters, rudders, and fins. The completed source code implemented in these nodes can be found in the IEC Application program disk. The file names are FX78_12.nc and FANCOIL.nc.

This project uses a SLTA/2 to connect the serial devices in the NPS AUV. The actual serial devices of interests are the sonars. However, for workbench testing purposes, this project uses a portable computer to simulate a serial device that sends and

receives serial data to the LonWorks network. This is a convenient diagnosis technique when testing any serial device. The SLTA/2 is a network interface that connects the serial devices to the LonWorks network via RS-232 serial interface. It is a preprogrammed device and its source files are included on the SLTA/2 software disk. The completed descriptions of those files are in the Echelon Serial LonTalk Adapter and Serial Gateway User's Guide.

D. SOURCE CODE DEVELOPMENT FOR THE INTEGRATED NETWORK ONBOARD AUV

This project uses the following hardware devices for the workbench testing:

- Emulator one is simulated as the command center
- An IEC Flexible I/O node controls two propeller motors
- An IEC Flexible I/O node controls two thruster motors
- An IEC Flexible I/O node controls two rudder motors
- Emulator two uses its light indicator to indicate the status of the propeller motors
- An SLTA/2 connects to a portable computer which simulates the sonar serial port communications

A sample source code has been developed and implemented into the emulator one node as the command center. The purpose of this source code is to demonstrate that the command center has the ability to control various devices onboard the NPS AUV by modifying many network variables in a LonWorks network environment. The source code is included in Appendix C. It has 15 different time frames. The command center issues many commands to different nodes. Those components then perform different tasks based on the commands they received in each time frame. Table 7.4 summarizes the functions of each time frame.

The first step of this network integration is to attach each node to the network communication media. Connections include one network manager, one protocol analyzer, two emulators, three IEC Flexible I/O nodes, and one SLTA/2. They are all connected to a twisted-pair bus topology network.

Next, the network manager recognizes and assigns each node's address by sequentially pushing each node's service pin when ready to be recognized. The node then sends out its unique 48-bit Neuron ID to the network manager. Table 7.1 describes the names of all nodes in this project. These node addresses are logical addresses that uniquely identify each the node in the network.

The LonWorks network's use of the logical addresses, rather than a physical serial number for each node has several advantages. First, a single message can be sent to multiple nodes without any confusion. Second, the network maintenance can be simplified. A new node can replace a damaged node without reconfiguring the entire network. The new node can be given the same network logical address and connection information as the damaged node. Therefore, the replacement of a physical device is apparent to hardware and software-controlled devices in the network.

After the network manager recognizes each node, it is required to queue each node's network variables. The network variables defined in Table 7.2 provide all interoperable communication between nodes in LonWorks network.

The final step of this project is to make desired logical connections using these network variables. This is called a binding process. Figure 7.3 shows the connection and directions of these network variables.

After the binding process, the LonBuilder performs an automatic build for the network (Echelon, LonBuilder User's Guide). When the automatic build process is completed and successful, the emulator with the sample source code is ready to simulate a maneuvering exercise for the workbench testing of the NPS AUV.

Time frame	Functions
Frame 1	Go forward straight at 20% of full speed
Frame 2	Change speed to 40% of full speed
Frame 3	Change speed to 80% of full speed
Frame 4	Change speed to full speed
Frame 5	Change speed to 80% of full speed
Frame 6	Change speed to 40% of full speed
Frame 7	Change speed to 20% of full speed
Frame 8	Stop and rudders amidships
Frame 9	Go forward straight at full speed
Frame 10	All stop
Frame 11	Turn right at 10% of full speed
Frame 12	Stop and rudders amidships
Frame 13	Turn left at 50% of full speed
Frame 14	Stop and rudders amidships
Frame 15	Go forward straight at 10% of full speed

Table 7.4 Time Frame for the AUV Maneuvering Exercise

E. SUMMARY

This chapter demonstrates the ease and flexibility of using LonWorks technology onboard the NPS AUV. There are numerous general-purpose application nodes that have been developed by third party companies. Nodes are off-the-shelf products with hardware ready to be used in a LonWorks network environment. Network integration is not a difficult task. The LonWorks network uses SNVTs to communicate among all nodes. The LonWorks network manager provides the binding process to all the network variables. Once all the nodes are logically connected by their network variables, a sample application program can run in the networked environment. The sample program described in this chapter makes a simulated AUV perform many different tasks using “when” clause statements for each time frame. These “when” clause statements are straightforward and simple, and only changes the values of the network variables to accomplish its desired mission.

VIII. CONCLUSIONS AND RECOMMENDATIONS

A. INTRODUCTION

Although this project demonstrated a method of simplifying the command and control of devices onboard *Phoenix* through the LonWorks networked control system, it represents only a starting point for further evaluation of the application of Echelon LonWorks technology onboard the NPS AUV. Many of the problems discussed in Chapter III have been solved by this approach but additional research and testing must be completed before a working model of LonWorks networked control system can be implemented within *Phoenix*.

B. RESEARCH CONCLUSIONS

1. Data Processing Rate Improvement

This project has developed a LonWorks networked control system with a bandwidth of 1.25 Mbps at peak rate which can handle a peak network traffic load of 1000 packet messages per second and a sustained continuous packet load can be 600-800 packet messages per second. Typical packet size is about 12 bytes.

The major component that provides this system with a bandwidth of 1.25 Mbps is the 10 MHz Neuron processor in each Neuron node. There are three 8-bit CPUs in each Neuron processor. This allows each Neuron node to collect and process all data locally. This added processing power from each Neuron node in the network system improves the overall network processing rate dramatically.

This project provides evidence that the LonWorks networked control system can solve the data rate problem (see Chapter III) and provide extensive future growth capability. This feature needs to be explicitly tested once all components are connected and operating.

2. Network Architecture Simplification

This project uses a simple bus-topology LonWorks networked control system. Two terminators are located at either end of a twisted pair bus lineup with multiple Neuron nodes in between. All Neuron nodes are configured to meet the interoperability

requirements of LonWorks technology before attaching to the network. External devices can then simply be plugged into nodes and are ready for operation.

The network architecture developed for this project simplifies the current architecture implemented onboard the NPS AUV. It reduces the amount of wiring required to connect the myriad of external devices to meet the operational requirements of the NPS AUV. The project successfully used the LonBuilder network manager to make logical connections between components without adding more wires.

3. No System Reconfiguration when Adding or Removing Devices

Every time a device is added or removed from the current system, a change to the entire software configuration is required. It is a long, tedious, and often frustrating experience to rework these configurations. Each Neuron node in the LonWorks network operates independently, and all external devices interact with a single Neuron node without affecting other nodes. There is no longer a need to perform a complete system reconfiguration when adding or removing devices. This greatly simplifies network maintenance and component upgrades.

4. Real-Time Response

This project has demonstrated that the new network system has higher data bandwidth, increased speed at each Neuron node, and a more scalable network architecture. This is a vast improvement over the current system onboard the NPS AUV. It has much greater potential to provide real-time data acquisition for vehicle operation.

5. Suitability for Other Robot Architectures

Control network technology provides rapid and expanded data processing capability for each Neuron node through an architecture that supports a large data bandwidth. The superior data bandwidth will expedite response for any large robot control system. Applying this technology to robot control systems can enhance coordination among all of the system's components since each component attached to a Neuron node is also provided additional local data processing power. This control network appears suitable for a variety of different robot types.

C. RECOMMENDATIONS FOR FUTURE WORK

This project provides a workbench implementation of selected portions the NPS AUV control system using the Echelon LonWorks technology. More work and research is needed before a working model can be installed in the actual vehicle for in-water testing and use.

1. Implement LonWorks to NPS AUV

This project only uses one of each kind of component on board the AUV to investigate the feasibility of using control network technology in the NPS *Phoenix* AUV. The workbench testing successfully demonstrated that LonWorks hardware and the LonTalk protocol can be implemented in the *Phoenix* to improve its current network system. This workbench setup needs expansion to include all necessary devices inside the AUV, listed in Table 7.3 and Figure 7.3. Previous chapters have described all of the essential steps and procedures required to build and complete a control network system. This new control network system should be implemented in the actual NPS AUV for its in-water testing.

2. TCP/IP-to-LonTalk Telemetry Bridge

The current execution level programming source code of *Phoenix* is written in C running in a TCP/IP networking environment. Because of the large amount of source code involved, a mapping between the TCP/IP protocol and the LonTalk protocol is a must. Otherwise the execution level source code may have to be rewritten in many pieces of Neuron C, the programming language for LonTalk. Such a partition is unlikely to be compatible with the RBM architecture. There are now several types of commercially available routers that can provide the connectivity to LonTalk from Ethernet or from the Internet using TCP/IP. A suitability test is needed to assess the feasibility of this technology for the telemetry between two different network protocols. Currently available router hardware may be too large to fit inside *Phoenix*. Detailed information about this technology is available in the references (Coactive 97).

D. SUMMARY

This project demonstrated the feasibility of using the LonWorks technology to implement a faster and more scalable networked control system onboard the NPS AUV. This technology has proven that it can provide reliable communication, decentralized (peer-to-peer) topology with no single point of failure, and easy extensibility and interoperability for a wide variety of hardware devices. It can greatly increase the reliability and throughput of *Phoenix* onboard sensors. This provides the required real-time data analysis capability needed for the *Phoenix's* missions. Implementing the Echelon LonWorks technology onboard the NPS AUV fulfills the ultimate goal of this thesis.

APPENDIX A- MASTER SNVTs LIST

This list provides all available SNVTs and details of their definitions. Standard Network Variable Types facilitate interoperability by providing a well-defined interface for communication between different Neuron nodes (The SNVT Master List and Programmer's Guide).

Measurement	Name	Range (Resolution)	SNVT #
Address, Neuron Chip	SNVT_address	0x4000 .. 0xF1FF (hexadecimal)	114
Alarm state	SNVT_alarm	see Structures below	88
Angular velocity	SNVT_angle_vel	-3,276.8 .. 3,276.7 radians/sec (0.1 radians/sec)	4
	SNVT_angle_vel_f	-1E38 .. 1E38 radians/sec	50
	SNVT_rpm ³	0 .. 65,534 <i>revolutions/minute</i> (1 rpm)	102
Area	SNVT_area ³	0 .. 13.1068 m ² (200 mm ²)	110
Character	SNVT_char_ascii	0 .. 255	7
Char string	SNVT_str_asc	see Structures below	36
	SNVT_str_int	see Structures below	37
Color	SNVT_color	see Structures below	70
Concentration	SNVT_ppm	0 .. 65,535 parts per million (1 ppm)	29
	SNVT_ppm_f	0 .. 1E38 ppm	58
Count, event	SNVT_count	0 .. 65,535 counts (1 count)	8
	SNVT_count_f	-1E38 .. 1E38 counts	51
Count, incremental	SNVT_count_inc	-32,768 .. 32,767 counts (1 count)	9
	SNVT_count_inc_f	-1E38 .. 1E38 counts	52
Currency	SNVT_currency	see Structures below	89
Current	SNVT_amp	-3,276.8 .. 3,276.7 amps (0.1 A)	1
	SNVT_amp_f	-1E38 .. 1E38 A	48
	SNVT_amp_mil	-3,276.8 .. 3,276.7 mA (0.1 mA)	2
Date	SNVT_date_cal	Use SNVT_timestamp instead	10
Day of week	SNVT_date_day	see Enum Lists below	11
Density	SNVT_density	0 .. 32,767.5 kg/m ³ (0.5 kg/m ³)	100
	SNVT_density_f	0 .. 1E38 kg/m ³	101
Emergency mode, HVAC	SNVT_hvac_emerg	see Enum Lists below	103
Energy, elec	SNVT_elec_kwh	0 .. 65,535 kilowatt-hour (1 kWh)	13
	SNVT_elec_whr	0 .. 6,553.5 watt-hours (0.1 WHR)	14
	SNVT_elec_whr_f	0 .. 1E38 watt-hour	68
Energy, thermal	SNVT_btu_f	-1E38 .. 1E38 btu	67
	SNVT_btu_kilo	0 .. 65,535 kilo btu	5
	SNVT_btu_mega	0 .. 65,535 mega btu	6
File position	SNVT_file_pos	see Structures below	90
File request	SNVT_file_req	see Structures below	73
File status	SNVT_file_status	see Structures below	74

Measurement	Name	Range (Resolution)	SNVT #
Flow	SNVT_flow ³	0 .. 65,534 liters/sec (1 l/sec)	15
	SNVT_flow_f	-1E38 .. 1E38 l/sec	53
	SNVT_flow_mil	0 .. 65,535 milliliters/sec (1 ml/sec)	16
Frequency	SNVT_freq_f	-1E38 .. 1E38 Hertz	75
	SNVT_freq_hz	0 .. 6553.5 Hz (0.1 Hz)	76
	SNVT_freq_kilohz	0 .. 6553.5 kHz (0.1 kHz)	77
	SNVT_freq_milhz	0 .. 6.5535 Hz (0.0001 Hz)	78
Gain	SNVT_muldiv	see Structures below	91
Grammage	SNVT_grammage	0 .. 6,553.5 gsm (0.1 gsm)	71
	SNVT_grammage_f	-1E38 .. 1E38 gsm	72
HVAC mode	SNVT_hvac_mode ²	see Enum Lists below	108
HVAC override	SNVT_hvac_overid ²	see Structures below	111
HVAC status	SNVT_hvac_status ²	see Structures below	112
Humidity	SNVT_lev_percent	See Level, percent below	81
Illumination	SNVT_lux	0 .. 65,535 lux (1 lux)	79
Installation source	SNVT_config_src	see Enum Lists below	69
Length	SNVT_length	0 .. 6,553.5 meters (0.1 m)	17
	SNVT_length_f	-1E38 .. 1E38 meters	54
	SNVT_length_kilo	0 .. 6,533.5 km (0.1 km)	18
	SNVT_length_micr	0 .. 6,553.5 μ m (0.1 μ m)	19
	SNVT_length_mil	0 .. 6,533.5 mm (0.1 mm)	20
	SNVT_lev_cont	0 .. 100 % (0.5%)	21
Level, continuous	SNVT_lev_cont_f	0 .. 100 %	55
	SNVT_lev_disc	see Enum Lists below	22
Level, percent	SNVT_lev_percent ⁴	-163.84% .. 163.83% (0.005% or 50 ppm)	81
Magnetic cards	SNVT_magcard	see Structures below	86
	SNVT_ISO_7811	Use SNVT_magcard instead	80
Mass	SNVT_mass	0 .. 6,553.5 grams (0.1 g)	23
	SNVT_mass_f	0 .. 1E38 g	56
	SNVT_mass_kilo	0 .. 6,553.5 kg (0.1 kg)	24
	SNVT_mass_mega	0 .. 6,553.5 metric tons (0.1 tonne)	25
	SNVT_mass_mil	0 .. 6,553.5 milligrams (0.1 mg)	26
	SNVT_multiplier	0 .. 32.7675 (0.0005)	82
Object request	SNVT_obj_request	see Structures below	92
Object status	SNVT_obj_status	see Structures below	93
Occupancy	SNVT_occupancy	see Enum Lists below	109
Override	SNVT_override	see Enum Lists below	97
Phase/rotation	SNVT_angle	0 .. 65.535 radians (0.001 radians)	3
	SNVT_angle_deg ⁴	-359.98 .. +360.00 degrees (0.02 degrees)	104
	SNVT_angle_f	-1E38 .. 1E38 radians	49
Phone state	SNVT_telcom	see Enum Lists below	38
Power	SNVT_power	0 .. 6,553.5 watts (0.1 W)	27
	SNVT_power_f	-1E38 .. 1E38 watts	57
	SNVT_power_kilo	0 .. 6,553.5 kW (0.1 kW)	28

Measurement	Name	Range (Resolution)	SNVT #
Power factor	SNVT_pwr_fact	-1.0 .. 1.0 (0.00005)	98
	SNVT_pwr_fact_f	-1.0 .. 1.0	99
Preset	SNVT_preset	see Structures below	94
Pressure - gauge	SNVT_press	-3,276.8 .. 3,276.7 kilopascals (0.1 kPa)	30
Pressure - absolute	SNVT_press_f	0 .. 1E38 pascals	59
Pressure - gauge	SNVT_press_p ⁴	-32,768 .. 32,766 pascals (1 Pa)	113
Resistance	SNVT_res	0 .. 6,553.5 ohms (0.1 Ω)	31
	SNVT_res_f	-1E38 .. 1E38 Ω	60
Sound level	SNVT_res_kilo	0 .. 6,553.5 k Ω (0.1 k Ω)	32
	SNVT_sound_db	-327.68 .. 327.67 decibels (0.01 dB)	33
	SNVT_sound_db_f	-1E38 .. 1E38 dBspl	61
Speed	SNVT_speed	0 .. 6,553.5 meters/sec (0.1 m/s)	34
	SNVT_speed_f	-1E38 .. 1E38 m/s	62
	SNVT_speed_mil	0 .. 65.535 m/s (0.001 m/s)	35
State	SNVT_state	see Structures below	83
Switch	SNVT_switch	see Structures below	95
Temperature	SNVT_temp ¹	-274 .. 6,279.5 °C (0.1 °C)	39
	SNVT_temp_p ^{2, 4}	-273.17 .. +327.66 °C (0.01 °C)	105
	SNVT_temp_f	-273.17 .. 1E38 °C	63
	SNVT_temp_setpt	see Structures below	106
Temperature setpts	SNVT_date_time	Use SNVT_timestamp instead	12
Time of day	SNVT_time_f	-1E38 .. 1E38 sec	64
Time - elapsed	SNVT_elapsed_tm	See Structures below	87
	SNVT_time_sec ³	0.0 .. 6553.4 sec (0.1 sec)	107
	SNVT_time_passed	Use SNVT_elapsed_tm instead	40
Time stamp	SNVT_time_stamp	see Structures below	84
Translation table	SNVT_trans_table	see Structures below	96
Volume	SNVT_vol	0 .. 6,553.5 liters (0.1 l)	41
	SNVT_vol_f	0 .. 1E38 l	65
	SNVT_vol_kilo	0 .. 6,553.5 kiloliters (0.1 kl)	42
	SNVT_vol_mil	0 .. 6,553.5 milliliters (0.1 ml)	43
	SNVT_volt	-3,276.8 .. 3,276.7 volts (0.1 V)	44
Voltage	SNVT_volt_dbmv	-327.68 .. 327.67 dB μ v (0.01 db μ v dc)	45
	SNVT_volt_f	-1E38 .. 1E38 volts	66
	SNVT_volt_kilo	-3,276.8 .. 3,276.7 kilovolts (0.1 kV)	46
	SNVT_volt_mil	-3,276.8 .. 3,276.7 millivolts (0.1 mV)	47
Zero and Span	SNVT_zerospans	see Structures below	85

¹ SNVT_temp represents tenths of a degree Celsius above -274 °C. To get SNVT_temp units define a constant: C_to_K equal to 2740 which is added to temperature expressed in tenths of degrees C.

² To be used for heating, ventilation and air conditioning applications.

³ The value 0xFFFF represents invalid data.

⁴ The value 0x7FFF represents invalid data.

APPENDIX B –ESTIMATED COST FOR THE NPS AUV USING LONWORKS TECHNOLOGY

Hardware Devices	Part Number	Educational discount Cost(50% off original cost)	Quantity	Function	Total Cost
Echelon LonBuilder Developer's kit	Model 20300	\$9,398.00	1	Network Development tools	\$9,398.00
IEC Flexible I/O Boards	None	\$450.00	6	Neuron Node for A/D D/A Devices (Servo amplifiers)	\$2,700.00
SLTA/2	730-00-1-310-1	\$270.00	3	Neuron Node for Serial Devices (Sonars)	\$810.00
PCLTA	731-00-11	\$158.00	1	PC to LonTalk Adapter for QNX Platform	\$158.00
RJ-45 Phone Jacks	None	\$7.00	12	Connectors in the Bus Topology	\$84.00
Terminators	None	\$10.00	2	Eliminate Unwanted Message Packets	\$20.00
Twist-pair wiring	None	\$20.00/50fts	1	Bus Wiring	\$20.00
Command Center (QNX PC)	None	\$1,300.00	1	Command Center	\$1,300.00
Personnel Training	None	\$1,000.00 per person	2		\$2,000
Total Cost					\$16,490

APPENDIX C – SOURCE CODE FOR THE AUV MANEUVERING EXERCISE

```

/*****
**
** File_name:  nav_plan.c
**
** Author:    Forrest Young
**
** Date:      December 20, 1997
**
** Purpose:   Provide a sample maneuvering exercise for the NPS AUV.
**            All networked control functions are provided by LonWorks
**            Neuron C language.
**
**
**
**
/*****
*****/

/*****
*****/

Compiler Pragmas
*****/

/* The following line contains the node's Self Documentation string. */

#pragma enable sd nv names

/*****
*****/

Include Files
*****/

#include <snvt_rq.h>
#include <snvt_lev.h>

/*****
*****/

Constant Declarations
*****/

unsigned int brightness = 0;

/*****
*****/

Input/Output Declarations
*****/

IO_4 input bit ioSwitch4;
IO_0 output pulsewidth clock (7) ioLed0;
```

```

/*****
Network Variables Declarations
*****/

```

```

network output SNVT_count nvo_l_prop;          /*left propeller*/
network output SNVT_count nvo_r_prop;          /*right propeller*/
network output SNVT_count nvo_l_rudder;        /*left rudder*/
network output SNVT_count nvo_r_rudder;        /*right rudder*/
network output SNVT_count nvo_l_thruster;      /*left thruster*/
network output SNVT_count nvo_r_thruster;      /*right thruster*/

network output SNVT_lev_disc nvo_l_indicator;  /* left indicator*/
network output SNVT_lev_disc nvo_r_indicator;  /* right indicator*/

```

```

/*****
Timer Declarations
*****/

```

```

mtimer frame1  = 2500;
mtimer frame2  = 5000;
mtimer frame3  = 7500;
mtimer frame4  = 10000;
mtimer frame5  = 12500;
mtimer frame6  = 15000;
mtimer frame7  = 17500;
mtimer frame8  = 20000;
mtimer frame9  = 22500;
mtimer frame10 = 25000;
mtimer frame11 = 27500;
mtimer frame12 = 30000;
mtimer frame13 = 32500;
mtimer frame14 = 35000;
mtimer frame15 = 37500;

```

```

/*****
Reset Task
*****/

```

```

when (reset)
{
    io_change_init (ioSwitch4);
    io_out (ioLed0,0);

    nvo_l_prop      = 0;          /*left propeller*/
    nvo_r_prop      = 0;          /*right propeller*/
    nvo_l_rudder    = 0;          /*left rudder*/
    nvo_r_rudder    = 0;          /*right rudder*/
    nvo_l_thruster  = 0;          /*left thruster*/
    nvo_r_thruster  = 0;          /*right thruster*/
}

```

```

        nvo_l_indicator = 1;          /* left indicator*/
        nvo_r_indicator = 1;          /* right indicator*/
    }

/*****
Frame1 task
Task:  direction:  Go forward straight
       speed:      20% of full speed
       Indication:  20% of full brightness
*****/

when (timer_expires(frame1))
{
    nvo_l_prop      = 800;
    nvo_r_prop      = 800;
    nvo_l_indicator = 52;
    nvo_r_indicator = 52;
    io_out (ioLed0, 52);
}

/*****
Frame2 task
Task:  direction:  Go forward straight
       speed:      40% of full speed
       Indication:  40% of full brightness
*****/

when (timer_expires(frame2))
{
    nvo_l_prop      = 1600;
    nvo_r_prop      = 1600;
    nvo_l_indicator = 104;
    nvo_r_indicator = 104;
    io_out (ioLed0, 104);
}

/*****
Frame3 task
Task:  direction:  Go forward straight
       speed:      80% of full speed
       Indication:  80% of full brightness
*****/

when (timer_expires(frame3))
{
    nvo_l_prop      = 3200;
    nvo_r_prop      = 3200;
    nvo_l_indicator = 204;
    nvo_r_indicator = 204;
    io_out (ioLed0, 204);
}

```

```

}

/*****
Frame4 task
Task:  direction:  Go forward straight
      speed:      full speed
      Indication:  full brightness
*****/

when (timer_expires(frame4))
{
    nvo_l_prop      = 4095;
    nvo_r_prop      = 4095;
    nvo_l_indicator = 255;
    nvo_r_indicator = 255;
    io_out (ioLed0, 255);
}

/*****
Frame5 task
Task:  direction:  Go forward straight
      speed:      80% of full speed
      Indication:  80% of full brightness
*****/

when (timer_expires(frame5))
{
    nvo_l_prop      = 3200;
    nvo_r_prop      = 3200;
    nvo_l_indicator = 204;
    nvo_r_indicator = 204;
    io_out (ioLed0, 204);
}

/*****
Frame6 task
Task:  direction:  Go forward straight
      speed:      40% of full speed
      Indication:  40% of full brightness
*****/

when (timer_expires(frame6))
{
    nvo_l_prop      = 1600;
    nvo_r_prop      = 1600;
    nvo_l_indicator = 104;
    nvo_r_indicator = 104;
    io_out (ioLed0, 104);
}

/*****
Frame7 task

```

```

Task:   direction:   Go forward straight
        speed:       20% of full speed
        Indication:  20% of full brightness
*****/

when (timer_expires(frame7))
{
    nvo_l_prop      = 800;
    nvo_r_prop      = 800;
    nvo_l_indicator = 52;
    nvo_r_indicator = 52;
    io_out (ioLed0, 52);
}

/*****
Frame8 task
Task:   direction:   idle
        speed:       stop
        Indication:  off
*****/

when (timer_expires(frame8))
{
    nvo_l_prop      = 0;
    nvo_r_prop      = 0;
    nvo_l_indicator = 0;
    nvo_r_indicator = 0;
    io_out (ioLed0, 0);
}

/*****
Frame9 task
Task:   direction:   Go forward straight
        speed:       full speed
        Indication:  full brightness
*****/

when (timer_expires(frame9))
{
    nvo_l_prop      = 4095;
    nvo_r_prop      = 4095;
    nvo_l_indicator = 255;
    nvo_r_indicator = 255;
    io_out (ioLed0, 255);
}

/*****
Frame10 task
Task:   direction:   idle
        speed:       stop
        Indication:  off
*****/

```

```

when (timer_expires(frame10))
{
    nvo_l_prop      = 0;
    nvo_r_prop      = 0;
    nvo_l_indicator = 0;
    nvo_r_indicator = 0;
    io_out (ioLed0, 0);
}

```

```

/*****
                                Framell task
Task:  direction:      turn right
       speed:         10% of full speed
       Indication:     10% of full brightness
*****/

```

```

when (timer_expires(frame11))
{
    nvo_l_prop      = 0;
    nvo_r_prop      = 400;
    nvo_l_thruster   = 0;
    nvo_r_thruster   = 0;
    nvo_l_rudder     = 128;
    nvo_r_rudder     = 255;
    nvo_l_indicator  = 0;
    nvo_r_indicator  = 26;
    io_out (ioLed0, 0);
}

```

```

/*****
                                Frame12 task
Task:  direction:      Stop and Point straight
       speed:          stop
       Indication:     off
*****/

```

```

when (timer_expires(frame12))
{
    nvo_l_prop      = 0;
    nvo_r_prop      = 0;
    nvo_l_thruster   = 0;
    nvo_r_thruster   = 0;
    nvo_l_rudder     = 128;
    nvo_r_rudder     = 128;
    nvo_l_indicator  = 0;
    nvo_r_indicator  = 0;
    io_out (ioLed0, 0);
}

```

```

/*****

```

```

                                Frame13 task
Task:  direction:      turn left
       speed:         50% of full speed
       Indication:     50% of full brightness
*****/

when (timer_expires(frame13))
{
    nvo_l_prop      = 2020;
    nvo_r_prop      = 0;
    nvo_l_thruster  = 0;
    nvo_r_thruster  = 0;
    nvo_l_rudder    = 255;
    nvo_r_rudder    = 128;
    nvo_l_indicator = 127;
    nvo_r_indicator = 0;
    io_out (ioLed0, 127);
}

/*****
                                Frame14 task
Task:  direction:      Stop and Point straight
       speed:         stop
       Indication:     off
*****/

when (timer_expires(frame14))
{
    nvo_l_prop      = 0;
    nvo_r_prop      = 0;
    nvo_l_thruster  = 0;
    nvo_r_thruster  = 0;
    nvo_l_rudder    = 128;
    nvo_r_rudder    = 128;
    nvo_l_indicator = 0;
    nvo_r_indicator = 0;
    io_out (ioLed0, 0);
}

/*****
                                Frame15 task
Task:  direction:      Go forward straight
       speed:         10% of full speed
       Indication:     10% of full brightness
*****/

when (timer_expires(frame15))
{
    nvo_l_prop      = 400;
    nvo_r_prop      = 400;
    nvo_l_indicator = 26;
    nvo_r_indicator = 26;

```



```

        io_out (ioLed0, 26);
    }

    /*****
        Input/Output event tasks
    Task:  When push io_4 switch, abort the mission
           Stop the vehicle
           Point direction straight ahead
           turn off both indicators
    *****/

    when (io_changes (ioSwitch4))
    {
        nvo_l_prop      = 0;
        nvo_r_prop      = 0;
        nvo_l_thruster  = 0;
        nvo_r_thruster  = 0;
        nvo_l_rudder    = 128;
        nvo_r_rudder    = 128;
        nvo_l_indicator = 0;
        nvo_r_indicator = 0;
        io_out (ioLed0, 0);
    }

    /*end of sample navigation exercise*/

```

LIST OF REFERENCES

Boorda, J. M., "Mine Countermeasures-An Integral Part Of Our Strategy And Our Forces," U.S. Navy, White Paper, December 1995.

Brutzman, Donald P., *From Virtual World to Reality: Designing an Autonomous Underwater Robot*, Proceedings of the Autonomous Vehicles in Mine Countermeasures Symposium, Naval Postgraduate School, Monterey, California, April, 1995.
Available at <http://www.cs.nps.navy.mil/research/auv>

Brutzman, Donald P., *NPS AUV Intergrated Simulator*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1992.

Brutzman, Don, Healey, Tony, Marco, Dave and McGhee, Bob, "The Phoenix Autonomous Underwater Vehicle," *AI-Based Mobile Robots*, editors David Kortenkamp, Pete Bonasso and Robin Murphy, MIT/AAAI Press, Cambridge Massachusetts, to appear 1998. Available at <http://www.stl.nps.navy.mil/~auv/aimr.ps>

Burns, Michael L., *Merging Virtual and Real Execution Level Control Software for the Phoenix Autonomous Underwater Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1996. Available at <http://www.cs.nps.navy.mil/research/auv>

Byrnes, R.B., *The Rational Behavior Model: A Multi-Paradigm, Tri-level Software Architecture for the Control of Autonomous Vehicles*, Ph.D. Dissertation, Naval Postgraduate School, Monterey, California, March 1993.

Campbell, Michael, *Real-Time Sonar Classification for Autonomous Underwater Vehicles*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1996. Available at <http://www.cs.nps.navy.mil/research/auv>

Coactive Aesthetics, Web Pages at <http://www.coactive.com>, Sausalito, California, 1997.

Echelon Corp., *An Introduction to the LonWorks Standard*, Palo Alto, California, 1997.
Available at <http://www.echelon.com/>

Echelon Corp., *LonBuilder Hardware Guide*, Palo Alto, California, 1995.

Echelon Corp., *LonBuilder User's Guide*, Palo Alto, California, 1995.

Echelon Corp., *LonWorks Product Manual*, Palo Alto, California, 1995.

Echelon Corp., *LonWorks Reference CD-ROM*, Palo Alto, California, 1997.
Available at <http://www.echelon.com/>

Echelon Corp., *Neuron C Programmer's Guide*, Palo Alto, California, 1995.

- Echelon Corp., *Neuron C Reference Guide*, Palo Alto, California, 1995.
- Echelon Corp., *Neuron Chip Data Book*, Palo Alto, California, 1995.
- Echelon Corp., *SNVT Master List and Programmer's Guide*, Palo Alto, California, March 1996.
- Echelon Corp., *LonWorks Training Manual*, Palo Alto, California, 1997
- Healey, A.J., *Mission Planning, Execution, and Data Analysis for the NPS AUV II Autonomous Underwater Vehicle*, Proceedings of the First IARP Workshop on Mobile Robots for Subsea Environments, Monterey, California, October, 1990.
- Intelligent Technologies Corp., *Flexible IO Users Manual*, Golden, Colorado, 1997.
- Leonhardt, Bradley, *Mission Planning and Mission Control Software for the Phoenix AUV: Implementation and Experimental Study*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1996. Available at <http://www.cs.nps.navy.mil/research/auv>
- LonMark Interoperability Association, *LonMark Application Layer Interoperability Guidelines*, Palo Alto, California, 1996.
- Marco, D. B., *Autonomous Control of Underwater Vehicles and Local Area Maneuvering*, Ph.D. Dissertation, Naval Postgraduate School, Monterey, California, September 1996. Available at <http://www.cs.nps.navy.mil/research/auv>
- Parallax Inc., Web pages at <http://www.parallaxinc.com>, Rocklin, California, 1997.
- Stallings, William, *Data and Computer Communications*, Prentice-Hall, Inc., 1997.
- Tritech International Ltd, *Tritech User Manual*, Mike E. Chapman Company, Duvall, Washington, 1992.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center..... 8725 John J. Kingman Road, Suite 0944 Ft. Belvoir, VA 22060-6218	2
2. Dudley Knox Library..... Naval Postgraduate School 411 Dyer Rd. Monterey, CA 93943-5101	2
3. Chairman, Code EC..... Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	1
4. Curr Officer, Code 37 CDR McMaster Naval Postgraduate School Monterey, CA 93943-5121	1
5. Dr. Xiaoping Yun, Code EC/Yx..... Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	2
6. Dr. Don Brutzman, Code UW/Br..... Computer Science Department Naval Postgraduate School Monterey, CA 93943-5118	2
7. Dr. Tony Healey, Code ME/Hy..... Mechanical Engineering Dept Naval Postgraduate School Monterey, CA 93943-5121	1
8. Dr. Robert B. McGhee, Code CS/Mz..... Computer Science Department Naval Postgraduate School Monterey, CA 93943-5118	1
9. Lt. Forrest Young..... 9638 La Capilla Ave Fountain Valley, CA 92708	1

10. CDR Mike J. Holden, USN, Code ME/Hm.....1
Mechanical Engineering Dept
Naval Postgraduate School
Monterey, CA 93943-5121
11. Dr. Jim Eagle, Chair, Code UW/Ea.....1
Naval Postgraduate School
Monterey, CA 93945-5101